Red Hat build of Node.js 10

Node.js Runtime Guide

Use Node.js 10 to develop scalable network applications that run on OpenShift and on stand-alone RHEL

Last Updated: 2020-07-01

Red Hat build of Node.js 10 Node.js Runtime Guide

Use Node.js 10 to develop scalable network applications that run on OpenShift and on stand-alone RHEL

Legal Notice

Abstract

This guide provides details on using the Node.js runtime.

Table of Contents

PREFACE

This guide covers concepts as well as practical details needed by developers to use the Node.js runtime.

CHAPTER 1. WHAT IS NODE.JS

Node.js is based on the V8 JavaScript engine from Google and allows you to write server-side JavaScript applications. It provides an I/O model based on events and non-blocking operations that enables you to write efficient applications. Node.js also provides a large module ecosystem called npm. Check out Additional Resources for further reading on Node.js.

The Node.js runtime enables you to run Node.js applications and services on OpenShift while providing all the advantages and conveniences of the OpenShift platform such as rolling updates, continuous delivery pipelines, service discovery, and canary deployments. OpenShift also makes it easier for your applications to implement common microservice patterns such as externalized configuration, health check, circuit breaker, and failover.

Red Hat provides different supported releases of Node.js. For more information how to get support, see Getting Node.js and support from Red Hat .

CHAPTER 2. SUPPORTED ARCHITECTURES BY NODE.JS

Node.js supports the following architectures:

- x86_64 (AMD64)
- IBM Z (s390x) in the OpenShift environment

Different images are supported for different architectures. The example codes in this guide demonstrate the commands for x86_64 architecture. If you are using other architectures, specify the relevant image name in the commands.

CHAPTER 3. INTRODUCTION TO EXAMPLE APPLICATIONS

Examples are working applications that demonstrate how to build cloud native applications and services. They demonstrate prescriptive architectures, design patterns, tools, and best practices that should be used when you develop your applications. The example applications can be used as templates to create your cloud-native microservices. You can update and redeploy these examples using the deployment process explained in this guide.

The examples implement Microservice patterns such as:

- Creating REST APIs
- Interoperating with a database
- Implementing the health check pattern
- Externalizing the configuration of your applications to make them more secure and easier to scale

You can use the examples applications as:

- Working demonstration of the technology
- Learning tool or a sandbox to understand how to develop applications for your project
- Starting point for updating or extending your own use case

Each example application is implemented in one or more runtimes. For example, the REST API Level 0 example is available for the following runtimes:

- Node.js
- Spring Boot
- Eclipse Vert.x
- Thorntail

The subsequent sections explain the example applications implemented for the Node.js runtime.

You can download and deploy all the example applications on:

- x86_64 architecture - The example applications in this guide demonstrate how to build and deploy example applications on x86_64 architecture.
- s390x architecture - To deploy the example applications on OpenShift environments provisioned on IBM Z infrastructure, specify the relevant IBM Z image name in the commands. Some of the example applications also require other products, such as Red Hat Data Grid to demonstrate the workflows. In this case, you must also change the image names of these products to their relevant IBM Z image names in the YAML file of the example applications.

CHAPTER 4. AVAILABLE EXAMPLES FOR NODE.JS

The Node.js runtime provides example applications. When you start developing applications on OpenShift, you can use the example applications as templates.

You can access these example applications on Developer Launcher.

4.1. REST API LEVEL 0 EXAMPLE FOR NODE.JS

IMPORTANT

The following example is not meant to be run in a production environment.

Example proficiency level: **Foundational**.

What the REST API Level 0 example does

The REST API Level 0 example shows how to map business operations to a remote procedure call endpoint over HTTP using a REST framework. This corresponds to Level 0 in the Richardson Maturity Model. Creating an HTTP endpoint using REST and its underlying principles to define your API lets you quickly prototype and design the API flexibly.

This example introduces the mechanics of interacting with a remote service using the HTTP protocol. It allows you to:

- Execute an HTTP **GET** request on the **api/greeting** endpoint.
- Receive a response in JSON format with a payload consisting of the **Hello, World!** String.
- Execute an HTTP **GET** request on the **api/greeting** endpoint while passing in a String argument. This uses the **name** request parameter in the query string.
- Receive a response in JSON format with a payload of **Hello, $name!** with **$name** replaced by the value of the **name** parameter passed into the request.

4.1.1. REST API Level 0 design tradeoffs

Table 4.1. Design tradeoffs

Pros	Cons

Pros	Cons
• The example application enables fast prototyping. • The API Design is flexible. • HTTP endpoints allow clients to be language-neutral.	• As an application or service matures, the REST API Level 0 approach might not scale well. It might not support a clean API design or use cases with database interactions. ◦ Any operations involving shared, mutable state must be integrated with an appropriate backing datastore. ◦ All requests handled by this API design are scoped only to the container servicing the request. Subsequent requests might not be served by the same container.

4.1.2. Deploying the REST API Level 0 example application to OpenShift Online

Use one of the following options to execute the REST API Level 0 example application on OpenShift Online.

- Use developers.redhat.com/launch
- Use the **oc** CLI client

Although each method uses the same **oc** commands to deploy your application, using developers.redhat.com/launch provides an automated deployment workflow that executes the **oc** commands for you.

4.1.2.1. Deploying the example application using developers.redhat.com/launch

Prerequisites

- An account at OpenShift Online.

Procedure

1. Navigate to the developers.redhat.com/launch URL in a browser and log in.
2. Follow on-screen instructions to create and launch your example application in Node.js.

4.1.2.2. Authenticating the oc CLI client

To work with example applications on OpenShift Online using the **oc** command-line client, you must authenticate the client using the token provided by the OpenShift Online web interface.

Prerequisites

- An account at OpenShift Online.

Procedure

1. Navigate to the OpenShift Online URL in a browser.

2. Click on the question mark icon in the top right-hand corner of the Web console, next to your user name.

3. Select *Command Line Tools* in the drop-down menu.

4. Find the text box that contains the **oc login …** command with the hidden token, and click the button next to it to copy its content to your clipboard.

5. Paste the command into a terminal application. The command uses your authentication token to authenticate your **oc** CLI client with your OpenShift Online account.

```
$ oc login OPENSHIFT_URL --token=MYTOKEN
```

4.1.2.3. Deploying the REST API Level 0 example application using the oc CLI client

Prerequisites

- The example application created using developers.redhat.com/launch. For more information, see Section 4.1.2.1, "Deploying the example application using developers.redhat.com/launch" .

- The **oc** client authenticated. For more information, see Section 4.1.2.2, "Authenticating the **oc** CLI client".

Procedure

1. Clone your project from GitHub.

```
$ git clone git@github.com:USERNAME/MY_PROJECT_NAME.git
```

   Alternatively, if you downloaded a ZIP file of your project, extract it.

```
$ unzip MY_PROJECT_NAME.zip
```

2. Create a new project in OpenShift.

```
$ oc new-project MY_PROJECT_NAME
```

3. Navigate to the root directory of your application.

4. Use **npm** to start the deployment to OpenShift.

```
$ npm install && npm run openshift
```

   These commands install any missing module dependencies, then using the Nodeshift module, deploy the example application on OpenShift.

5. Check the status of your application and ensure your pod is running.

```
$ oc get pods -w
NAME                      READY    STATUS     RESTARTS   AGE
MY_APP_NAME-1-aaaaa       1/1      Running    0          58s
```

```
MY_APP_NAME-s2i-1-build        0/1       Completed   0          2m
```

The **MY_APP_NAME-1-aaaaa** pod should have a status of **Running** once it is fully deployed and started. Your specific pod name will vary. The number in the middle will increase with each new build. The letters at the end are generated when the pod is created.

6. After your example application is deployed and started, determine its route.

   Example Route Information

   ```
   $ oc get routes
   NAME              HOST/PORT                                    PATH      SERVICES
   PORT      TERMINATION
   MY_APP_NAME       MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME
   MY_APP_NAME      8080
   ```

   The route information of a pod gives you the base URL which you use to access it. In the example above, you would use **http://MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME** as the base URL to access the application.

4.1.3. Deploying the REST API Level 0 example application to Minishift or CDK

Use one of the following options to execute the REST API Level 0 example application locally on Minishift or CDK:

- Using Fabric8 Launcher
- Using the **oc** CLI client

Although each method uses the same **oc** commands to deploy your application, using Fabric8 Launcher provides an automated deployment workflow that executes the **oc** commands for you.

4.1.3.1. Getting the Fabric8 Launcher tool URL and credentials

You need the Fabric8 Launcher tool URL and user credentials to create and deploy example applications on Minishift or CDK. This information is provided when the Minishift or CDK is started.

Prerequisites

- The Fabric8 Launcher tool installed, configured, and running.

Procedure

1. Navigate to the console where you started Minishift or CDK.

2. Check the console output for the URL and user credentials you can use to access the running Fabric8 Launcher:

   Example Console Output from a Minishift or CDK Startup

   ```
   ...
   -- Removing temporary directory ... OK
   -- Server Information ...
      OpenShift server started.
   ```

```
The server is accessible via web console at:
    https://192.168.42.152:8443

You are logged in as:
    User:     developer
    Password: developer

To login as administrator:
    oc login -u system:admin
```

4.1.3.2. Deploying the example application using the Fabric8 Launcher tool

Prerequisites

- The URL of your running Fabric8 Launcher instance and the user credentials of your Minishift or CDK. For more information, see Section 4.1.3.1, “Getting the Fabric8 Launcher tool URL and credentials”.

Procedure

1. Navigate to the Fabric8 Launcher URL in a browser.
2. Follow the on-screen instructions to create and launch your example application in Node.js.

4.1.3.3. Authenticating the oc CLI client

To work with example applications on Minishift or CDK using the **oc** command-line client, you must authenticate the client using the token provided by the Minishift or CDK web interface.

Prerequisites

- The URL of your running Fabric8 Launcher instance and the user credentials of your Minishift or CDK. For more information, see Section 4.1.3.1, “Getting the Fabric8 Launcher tool URL and credentials”.

Procedure

1. Navigate to the Minishift or CDK URL in a browser.
2. Click on the question mark icon in the top right-hand corner of the Web console, next to your user name.
3. Select *Command Line Tools* in the drop-down menu.
4. Find the text box that contains the **oc login …** command with the hidden token, and click the button next to it to copy its content to your clipboard.
5. Paste the command into a terminal application. The command uses your authentication token to authenticate your **oc** CLI client with your Minishift or CDK account.

```
$ oc login OPENSHIFT_URL --token=MYTOKEN
```

4.1.3.4. Deploying the REST API Level 0 example application using the oc CLI client

Prerequisites

- The example application created using Fabric8 Launcher tool on a Minishift or CDK. For more information, see Section 4.1.3.2, “Deploying the example application using the Fabric8 Launcher tool”.
- Your Fabric8 Launcher tool URL.
- The **oc** client authenticated. For more information, see Section 4.1.3.3, “Authenticating the **oc** CLI client”.

Procedure

1. Clone your project from GitHub.

   ```
   $ git clone git@github.com:USERNAME/MY_PROJECT_NAME.git
   ```

   Alternatively, if you downloaded a ZIP file of your project, extract it.

   ```
   $ unzip MY_PROJECT_NAME.zip
   ```

2. Create a new project in OpenShift.

   ```
   $ oc new-project MY_PROJECT_NAME
   ```

3. Navigate to the root directory of your application.
4. Use **npm** to start the deployment to OpenShift.

   ```
   $ npm install && npm run openshift
   ```

   These commands install any missing module dependencies, then using the Nodeshift module, deploy the example application on OpenShift.

5. Check the status of your application and ensure your pod is running.

   ```
   $ oc get pods -w
   NAME                         READY     STATUS      RESTARTS   AGE
   MY_APP_NAME-1-aaaaa              1/1       Running     0          58s
   MY_APP_NAME-s2i-1-build          0/1       Completed   0          2m
   ```

   The **MY_APP_NAME-1-aaaaa** pod should have a status of **Running** once it is fully deployed and started. Your specific pod name will vary. The number in the middle will increase with each new build. The letters at the end are generated when the pod is created.

6. After your example application is deployed and started, determine its route.

   #### Example Route Information

   ```
   $ oc get routes
   NAME           HOST/PORT                                              PATH      SERVICES
   PORT      TERMINATION
   MY_APP_NAME        MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME
   MY_APP_NAME      8080
   ```

The route information of a pod gives you the base URL which you use to access it. In the example above, you would use **http://MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME** as the base URL to access the application.

4.1.4. Deploying the REST API Level 0 example application to OpenShift Container Platform

The process of creating and deploying example applications to OpenShift Container Platform is similar to OpenShift Online:

Prerequisites

- The example application created using developers.redhat.com/launch.

Procedure

- Follow the instructions in Section 4.1.2, “Deploying the REST API Level 0 example application to OpenShift Online”, only use the URL and user credentials from the OpenShift Container Platform Web Console.

4.1.5. Interacting with the unmodified REST API Level 0 example application for Node.js

The example provides a default HTTP endpoint that accepts GET requests.

Prerequisites

- Your application running
- The **curl** binary or a web browser

Procedure

1. Use **curl** to execute a **GET** request against the example. You can also use a browser to do this.

```
$ curl http://MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME/api/greeting
{"content":"Hello, World!"}
```

2. Use **curl** to execute a **GET** request with the **name** URL parameter against the example. You can also use a browser to do this.

```
$ curl http://MY_APP_NAME-
MY_PROJECT_NAME.OPENSHIFT_HOSTNAME/api/greeting?name=Sarah
{"content":"Hello, Sarah!"}
```

NOTE

From a browser, you can also use a form provided by the example to perform these same interactions. The form is located at the root of the project **http://MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME**.

4.1.6. REST resources

More background and related information on REST can be found here:

- Architectural Styles and the Design of Network-based Software Architectures - Representational State Transfer (REST)
- Richardson Maturity Model
- Express Web Framework
- REST API Level 0 for Spring Boot
- REST API Level 0 for Eclipse Vert.x
- REST API Level 0 for Thorntail

4.2. EXTERNALIZED CONFIGURATION EXAMPLE FOR NODE.JS

IMPORTANT

The following example is not meant to be run in a production environment.

Example proficiency level: **Foundational**.

Externalized Configuration provides a basic example of using a ConfigMap to externalize configuration. *ConfigMap* is an object used by OpenShift to inject configuration data as simple key and value pairs into one or more Linux containers while keeping the containers independent of OpenShift.

This example shows you how to:

- Set up and configure a **ConfigMap**.
- Use the configuration provided by the **ConfigMap** within an application.
- Deploy changes to the **ConfigMap** configuration of running applications.

4.2.1. The externalized configuration design pattern

Whenever possible, externalize the application configuration and separate it from the application code. This allows the application configuration to change as it moves through different environments, but leaves the code unchanged. Externalizing the configuration also keeps sensitive or internal information out of your code base and version control. Many languages and application servers provide environment variables to support externalizing an application’s configuration.

Microservices architectures and multi-language (polyglot) environments add a layer of complexity to managing an application’s configuration. Applications consist of independent, distributed services, and each can have its own configuration. Keeping all configuration data synchronized and accessible creates a maintenance challenge.

ConfigMaps enable the application configuration to be externalized and used in individual Linux containers and pods on OpenShift. You can create a ConfigMap object in a variety of ways, including using a YAML file, and inject it into the Linux container. ConfigMaps also allow you to group and scale sets of configuration data. This lets you configure a large number of environments beyond the basic *Development*, *Stage*, and *Production*. You can find more information about ConfigMaps in the OpenShift documentation.

4.2.2. Externalized Configuration design tradeoffs

Table 4.2. Design Tradeoffs

Pros	Cons
• Configuration is separate from deployments • Can be updated independently • Can be shared across services	• Adding configuration to environment requires additional step • Has to be maintained separately • Requires coordination beyond the scope of a service

4.2.3. Deploying the Externalized Configuration example application to OpenShift Online

Use one of the following options to execute the Externalized Configuration example application on OpenShift Online.

- Use developers.redhat.com/launch
- Use the **oc** CLI client

Although each method uses the same **oc** commands to deploy your application, using developers.redhat.com/launch provides an automated deployment workflow that executes the **oc** commands for you.

4.2.3.1. Deploying the example application using developers.redhat.com/launch

Prerequisites

- An account at OpenShift Online.

Procedure

1. Navigate to the developers.redhat.com/launch URL in a browser and log in.
2. Follow on-screen instructions to create and launch your example application in Node.js.

4.2.3.2. Authenticating the oc CLI client

To work with example applications on OpenShift Online using the **oc** command-line client, you must authenticate the client using the token provided by the OpenShift Online web interface.

Prerequisites

- An account at OpenShift Online.

Procedure

1. Navigate to the OpenShift Online URL in a browser.

2. Click on the question mark icon in the top right-hand corner of the Web console, next to your user name.

3. Select *Command Line Tools* in the drop-down menu.

4. Find the text box that contains the **oc login …** command with the hidden token, and click the button next to it to copy its content to your clipboard.

5. Paste the command into a terminal application. The command uses your authentication token to authenticate your **oc** CLI client with your OpenShift Online account.

```
$ oc login OPENSHIFT_URL --token=MYTOKEN
```

4.2.3.3. Deploying the Externalized Configuration example application using the oc CLI client

Prerequisites

- The example application created using developers.redhat.com/launch. For more information, see Section 4.2.3.1, "Deploying the example application using developers.redhat.com/launch" .
- The **oc** client authenticated. For more information, see Section 4.2.3.2, "Authenticating the **oc** CLI client".

Procedure

1. Clone your project from GitHub.

   ```
   $ git clone git@github.com:USERNAME/MY_PROJECT_NAME.git
   ```

   Alternatively, if you downloaded a ZIP file of your project, extract it.

   ```
   $ unzip MY_PROJECT_NAME.zip
   ```

2. Create a new OpenShift project.

   ```
   $ oc new-project MY_PROJECT_NAME
   ```

3. Assign view access rights to the service account before deploying your example application, so that the application can access the OpenShift API in order to read the contents of the ConfigMap.

   ```
   $ oc policy add-role-to-user view -n $(oc project -q) -z default
   ```

4. Navigate to the root directory of your application.

5. Deploy your ConfigMap configuration to OpenShift using **app-config.yml**.

   ```
   $ oc create configmap app-config --from-file=app-config.yml
   ```

6. Verify your ConfigMap configuration has been deployed.

   ```
   $ oc get configmap app-config -o yaml
   ```

```
apiVersion: v1
data:
  app-config.yml: |-
      message : "Hello, %s from a ConfigMap !"
      level : INFO
...
```

7. Use **npm** to start the deployment to OpenShift.

   ```
   $ npm install && npm run openshift
   ```

   These commands install any missing module dependencies, then using the Nodeshift module, deploy the example application on OpenShift.

8. Check the status of your application and ensure your pod is running.

   ```
   $ oc get pods -w
   NAME                             READY     STATUS      RESTARTS   AGE
   MY_APP_NAME-1-aaaaa               1/1       Running     0          58s
   MY_APP_NAME-s2i-1-build           0/1       Completed   0          2m
   ```

   The **MY_APP_NAME-1-aaaaa** pod should have a status of **Running** once its fully deployed and started. Your specific pod name will vary. The number in the middle will increase with each new build. The letters at the end are generated when the pod is created.

9. After your example application is deployed and started, determine its route.

   Example Route Information

   ```
   $ oc get routes
   NAME           HOST/PORT                                        PATH      SERVICES
   PORT      TERMINATION
   MY_APP_NAME       MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME
   MY_APP_NAME      8080
   ```

   The route information of a pod gives you the base URL which you use to access it. In the example above, you would use **http://MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME** as the base URL to access the application.

4.2.4. Deploying the Externalized Configuration example application to Minishift or CDK

Use one of the following options to execute the Externalized Configuration example application locally on Minishift or CDK:

- Using Fabric8 Launcher
- Using the **oc** CLI client

Although each method uses the same **oc** commands to deploy your application, using Fabric8 Launcher provides an automated deployment workflow that executes the **oc** commands for you.

4.2.4.1. Getting the Fabric8 Launcher tool URL and credentials

You need the Fabric8 Launcher tool URL and user credentials to create and deploy example applications on Minishift or CDK. This information is provided when the Minishift or CDK is started.

Prerequisites

- The Fabric8 Launcher tool installed, configured, and running.

Procedure

1. Navigate to the console where you started Minishift or CDK.
2. Check the console output for the URL and user credentials you can use to access the running Fabric8 Launcher:

Example Console Output from a Minishift or CDK Startup

```
...
-- Removing temporary directory ... OK
-- Server Information ...
   OpenShift server started.
   The server is accessible via web console at:
       https://192.168.42.152:8443

   You are logged in as:
       User:     developer
       Password: developer

   To login as administrator:
       oc login -u system:admin
```

4.2.4.2. Deploying the example application using the Fabric8 Launcher tool

Prerequisites

- The URL of your running Fabric8 Launcher instance and the user credentials of your Minishift or CDK. For more information, see Section 4.2.4.1, “Getting the Fabric8 Launcher tool URL and credentials”.

Procedure

1. Navigate to the Fabric8 Launcher URL in a browser.
2. Follow the on-screen instructions to create and launch your example application in Node.js.

4.2.4.3. Authenticating the oc CLI client

To work with example applications on Minishift or CDK using the **oc** command-line client, you must authenticate the client using the token provided by the Minishift or CDK web interface.

Prerequisites

- The URL of your running Fabric8 Launcher instance and the user credentials of your Minishift or CDK. For more information, see Section 4.2.4.1, “Getting the Fabric8 Launcher tool URL and credentials”.

Procedure

1. Navigate to the Minishift or CDK URL in a browser.
2. Click on the question mark icon in the top right-hand corner of the Web console, next to your user name.
3. Select *Command Line Tools* in the drop-down menu.
4. Find the text box that contains the **oc login …** command with the hidden token, and click the button next to it to copy its content to your clipboard.
5. Paste the command into a terminal application. The command uses your authentication token to authenticate your **oc** CLI client with your Minishift or CDK account.

   ```
   $ oc login OPENSHIFT_URL --token=MYTOKEN
   ```

4.2.4.4. Deploying the Externalized Configuration example application using the oc CLI client

Prerequisites

- The example application created using Fabric8 Launcher tool on a Minishift or CDK. For more information, see Section 4.2.4.2, "Deploying the example application using the Fabric8 Launcher tool".
- Your Fabric8 Launcher tool URL.
- The **oc** client authenticated. For more information, see Section 4.2.4.3, "Authenticating the **oc** CLI client".

Procedure

1. Clone your project from GitHub.

   ```
   $ git clone git@github.com:USERNAME/MY_PROJECT_NAME.git
   ```

   Alternatively, if you downloaded a ZIP file of your project, extract it.

   ```
   $ unzip MY_PROJECT_NAME.zip
   ```

2. Create a new OpenShift project.

   ```
   $ oc new-project MY_PROJECT_NAME
   ```

3. Assign view access rights to the service account before deploying your example application, so that the application can access the OpenShift API in order to read the contents of the ConfigMap.

   ```
   $ oc policy add-role-to-user view -n $(oc project -q) -z default
   ```

4. Navigate to the root directory of your application.
5. Deploy your ConfigMap configuration to OpenShift using **app-config.yml**.

```
$ oc create configmap app-config --from-file=app-config.yml
```

6. Verify your ConfigMap configuration has been deployed.

```
$ oc get configmap app-config -o yaml

apiVersion: v1
data:
  app-config.yml: |-
      message : "Hello, %s from a ConfigMap !"
      level : INFO
...
```

7. Use **npm** to start the deployment to OpenShift.

```
$ npm install && npm run openshift
```

   These commands install any missing module dependencies, then using the Nodeshift module, deploy the example application on OpenShift.

8. Check the status of your application and ensure your pod is running.

```
$ oc get pods -w
NAME                             READY     STATUS      RESTARTS   AGE
MY_APP_NAME-1-aaaaa               1/1       Running     0          58s
MY_APP_NAME-s2i-1-build           0/1       Completed   0          2m
```

   The **MY_APP_NAME-1-aaaaa** pod should have a status of **Running** once its fully deployed and started. Your specific pod name will vary. The number in the middle will increase with each new build. The letters at the end are generated when the pod is created.

9. After your example application is deployed and started, determine its route.

   Example Route Information

```
$ oc get routes
NAME              HOST/PORT                                   PATH      SERVICES
PORT      TERMINATION
MY_APP_NAME         MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME
MY_APP_NAME      8080
```

   The route information of a pod gives you the base URL which you use to access it. In the example above, you would use **http://MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME** as the base URL to access the application.

4.2.5. Deploying the Externalized Configuration example application to OpenShift Container Platform

The process of creating and deploying example applications to OpenShift Container Platform is similar to OpenShift Online:

Prerequisites

- The example application created using developers.redhat.com/launch.

Procedure

- Follow the instructions in Section 4.2.3, "Deploying the Externalized Configuration example application to OpenShift Online", only use the URL and user credentials from the OpenShift Container Platform Web Console.

4.2.6. Interacting with the unmodified Externalized Configuration example application for Node.js

The example provides a default HTTP endpoint that accepts **GET** requests.

Prerequisites

- Your application running
- The **curl** binary or a web browser

Procedure

1. Use **curl** to execute a **GET** request against the example. You can also use a browser to do this.

   ```
   $ curl http://MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME/api/greeting
   {"content":"Hello, World from a ConfigMap !"}
   ```

2. Update the deployed ConfigMap configuration.

   ```
   $ oc edit configmap app-config
   ```

   Change the value for the **message** key to **Bonjour, %s from a ConfigMap !** and save the file.

3. Update of the ConfigMap should be read by the application within an acceptable time (a few seconds) without requiring a restart of the application.

4. Execute a **GET** request using **curl** against the example with the updated ConfigMap configuration to see your updated greeting. You can also do this from your browser using the web form provided by the application.

   ```
   $ curl http://MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME/api/greeting
   {"content":"Bonjour, World from a ConfigMap !"}
   ```

4.2.7. Externalized Configuration resources

More background and related information on Externalized Configuration and ConfigMap can be found here:

- OpenShift ConfigMap Documentation
- Blog Post about ConfigMap in OpenShift
- Externalized Configuration for Spring Boot
- Externalized Configuration for Eclipse Vert.x

- Externalized Configuration for Thorntail

4.3. RELATIONAL DATABASE BACKEND EXAMPLE FOR NODE.JS

IMPORTANT

The following example is not meant to be run in a production environment.

Limitation: Run this example application on a Minishift or CDK. You can also use a manual workflow to deploy this example to OpenShift Online Pro and OpenShift Container Platform. This example is not currently available on OpenShift Online Starter.

Example proficiency level: **Foundational**.

What the Relational Database Backend example does

The Relational Database Backend example expands on the REST API Level 0 application to provide a basic example of performing *create*, *read*, *update* and *delete* (*CRUD*) operations on a PostgreSQL database using a simple HTTP API. *CRUD* operations are the four basic functions of persistent storage, widely used when developing an HTTP API dealing with a database.

The example also demonstrates the ability of the HTTP application to locate and connect to a database in OpenShift. Each runtime shows how to implement the connectivity solution best suited in the given case. The runtime can choose between options such as using *JDBC*, *JPA*, or accessing *ORM* APIs directly.

The example application exposes an HTTP API, which provides endpoints that allow you to manipulate data by performing *CRUD* operations over HTTP. The *CRUD* operations are mapped to HTTP **Verbs**. The API uses JSON formatting to receive requests and return responses to the user. The user can also use a user interface provided by the example to use the application. Specifically, this example provides an application that allows you to:

- Navigate to the application web interface in your browser. This exposes a simple website allowing you to perform *CRUD* operations on the data in the **my_data** database.
- Execute an HTTP **GET** request on the **api/fruits** endpoint.
- Receive a response formatted as a JSON array containing the list of all fruits in the database.
- Execute an HTTP **GET** request on the **api/fruits/*** endpoint while passing in a valid item ID as an argument.
- Receive a response in JSON format containing the name of the fruit with the given ID. If no item matches the specified ID, the call results in an HTTP error 404.
- Execute an HTTP **POST** request on the **api/fruits** endpoint passing in a valid **name** value to create a new entry in the database.
- Execute an HTTP **PUT** request on the **api/fruits/*** endpoint passing in a valid ID and a name as an argument. This updates the name of the item with the given ID to match the name specified in your request.
- Execute an HTTP **DELETE** request on the **api/fruits/*** endpoint, passing in a valid ID as an argument. This removes the item with the specified ID from the database and returns an HTTP code **204** (No Content) as a response. If you pass in an invalid ID, the call results in an HTTP error **404**.

This example does not showcase a fully matured RESTful model (level 3), but it does use compatible HTTP verbs and status, following the recommended HTTP API practices.

4.3.1. Relational Database Backend design tradeoffs

Table 4.3. Design Tradeoffs

Pros	Cons
• Each runtime determines how to implement the database interactions. One can use a low-level connectivity API such as JDBC, some other can use JPA, and yet another can access ORM APIs directly. Each runtime decides what would be the best way. • Each runtime determines how the schema is created.	• The PostgreSQL database provided with this example application is not backed up with persistent storage. Changes to the database are lost if you stop or redeploy the database pod. To use an external database with your example application's pod in order to preserve changes, see the Creating an application with a database chapter of the OpenShift Documentation. It is also possible to set up persistent storage with database containers on OpenShift. (For more details about using persistent storage with OpenShift and containers, see the Persistent Storage, Managing Volumes and Persistent Volumes chapters of the OpenShift Documentation).

4.3.2. Deploying the Relational Database Backend example application to OpenShift Online

Use one of the following options to execute the Relational Database Backend example application on OpenShift Online.

- Use developers.redhat.com/launch
- Use the **oc** CLI client

Although each method uses the same **oc** commands to deploy your application, using developers.redhat.com/launch provides an automated deployment workflow that executes the **oc** commands for you.

4.3.2.1. Deploying the example application using developers.redhat.com/launch

Prerequisites

- An account at OpenShift Online.

Procedure

1. Navigate to the developers.redhat.com/launch URL in a browser and log in.
2. Follow on-screen instructions to create and launch your example application in Node.js.

4.3.2.2. Authenticating the oc CLI client

To work with example applications on OpenShift Online using the **oc** command-line client, you must authenticate the client using the token provided by the OpenShift Online web interface.

Prerequisites

- An account at OpenShift Online.

Procedure

1. Navigate to the OpenShift Online URL in a browser.
2. Click on the question mark icon in the top right-hand corner of the Web console, next to your user name.
3. Select *Command Line Tools* in the drop-down menu.
4. Find the text box that contains the **oc login …** command with the hidden token, and click the button next to it to copy its content to your clipboard.
5. Paste the command into a terminal application. The command uses your authentication token to authenticate your **oc** CLI client with your OpenShift Online account.

```
$ oc login OPENSHIFT_URL --token=MYTOKEN
```

4.3.2.3. Deploying the Relational Database Backend example application using the oc CLI client

Prerequisites

- The example application created using developers.redhat.com/launch. For more information, see Section 4.3.2.1, "Deploying the example application using developers.redhat.com/launch" .
- The **oc** client authenticated. For more information, see Section 4.3.2.2, "Authenticating the **oc** CLI client".

Procedure

1. Clone your project from GitHub.

   ```
   $ git clone git@github.com:USERNAME/MY_PROJECT_NAME.git
   ```

   Alternatively, if you downloaded a ZIP file of your project, extract it.

   ```
   $ unzip MY_PROJECT_NAME.zip
   ```

2. Create a new OpenShift project.

   ```
   $ oc new-project MY_PROJECT_NAME
   ```

3. Navigate to the root directory of your application.
4. Deploy the PostgreSQL database to OpenShift. Ensure that you use the following values for user name, password, and database name when creating your database application. The

example application is pre-configured to use these values. Using different values prevents your application from integrating with the database.

```
$ oc new-app -e POSTGRESQL_USER=luke -ePOSTGRESQL_PASSWORD=secret -
ePOSTGRESQL_DATABASE=my_data registry.access.redhat.com/rhscl/postgresql-10-rhel7
--name=my-database
```

5. Check the status of your database and ensure the pod is running.

```
$ oc get pods -w
my-database-1-aaaaa   1/1       Running   0         45s
my-database-1-deploy   0/1       Completed   0         53s
```

The **my-database-1-aaaaa** pod should have a status of **Running** and should be indicated as ready once it is fully deployed and started. Your specific pod name will vary. The number in the middle will increase with each new build. The letters at the end are generated when the pod is created.

6. Use **npm** to start the deployment to OpenShift.

```
$ npm install && npm run openshift
```

These commands install any missing module dependencies, then using the Nodeshift module, deploy the example application on OpenShift.

7. Check the status of your application and ensure your pod is running.

```
$ oc get pods -w
NAME                             READY     STATUS      RESTARTS   AGE
MY_APP_NAME-1-aaaaa       1/1       Running     0          58s
MY_APP_NAME-s2i-1-build   0/1       Completed   0          2m
```

Your **MY_APP_NAME-1-aaaaa** pod should have a status of **Running** and should be indicated as ready once it is fully deployed and started.

8. After your example application is deployed and started, determine its route.

Example Route Information

```
$ oc get routes
NAME              HOST/PORT                                PATH      SERVICES              PORT
TERMINATION
MY_APP_NAME   MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME
MY_APP_NAME   8080
```

The route information of a pod gives you the base URL which you use to access it. In the example above, you would use **http://MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME** as the base URL to access the application.

4.3.3. Deploying the Relational Database Backend example application to Minishift or CDK

Use one of the following options to execute the Relational Database Backend example application locally on Minishift or CDK:

- Using Fabric8 Launcher
- Using the **oc** CLI client

Although each method uses the same **oc** commands to deploy your application, using Fabric8 Launcher provides an automated deployment workflow that executes the **oc** commands for you.

4.3.3.1. Getting the Fabric8 Launcher tool URL and credentials

You need the Fabric8 Launcher tool URL and user credentials to create and deploy example applications on Minishift or CDK. This information is provided when the Minishift or CDK is started.

Prerequisites

- The Fabric8 Launcher tool installed, configured, and running.

Procedure

1. Navigate to the console where you started Minishift or CDK.
2. Check the console output for the URL and user credentials you can use to access the running Fabric8 Launcher:

   Example Console Output from a Minishift or CDK Startup

```
...
-- Removing temporary directory ... OK
-- Server Information ...
   OpenShift server started.
   The server is accessible via web console at:
       https://192.168.42.152:8443

   You are logged in as:
       User:     developer
       Password: developer

   To login as administrator:
       oc login -u system:admin
```

4.3.3.2. Deploying the example application using the Fabric8 Launcher tool

Prerequisites

- The URL of your running Fabric8 Launcher instance and the user credentials of your Minishift or CDK. For more information, see Section 4.3.3.1, “Getting the Fabric8 Launcher tool URL and credentials”.

Procedure

1. Navigate to the Fabric8 Launcher URL in a browser.
2. Follow the on-screen instructions to create and launch your example application in Node.js.

4.3.3.3. Authenticating the oc CLI client

To work with example applications on Minishift or CDK using the **oc** command-line client, you must authenticate the client using the token provided by the Minishift or CDK web interface.

Prerequisites

- The URL of your running Fabric8 Launcher instance and the user credentials of your Minishift or CDK. For more information, see Section 4.3.3.1, "Getting the Fabric8 Launcher tool URL and credentials".

Procedure

1. Navigate to the Minishift or CDK URL in a browser.
2. Click on the question mark icon in the top right-hand corner of the Web console, next to your user name.
3. Select *Command Line Tools* in the drop-down menu.
4. Find the text box that contains the **oc login …** command with the hidden token, and click the button next to it to copy its content to your clipboard.
5. Paste the command into a terminal application. The command uses your authentication token to authenticate your **oc** CLI client with your Minishift or CDK account.

   ```
   $ oc login OPENSHIFT_URL --token=MYTOKEN
   ```

4.3.3.4. Deploying the Relational Database Backend example application using the **oc** CLI client

Prerequisites

- The example application created using Fabric8 Launcher tool on a Minishift or CDK. For more information, see Section 4.3.3.2, "Deploying the example application using the Fabric8 Launcher tool".
- Your Fabric8 Launcher tool URL.
- The **oc** client authenticated. For more information, see Section 4.3.3.3, "Authenticating the **oc** CLI client".

Procedure

1. Clone your project from GitHub.

   ```
   $ git clone git@github.com:USERNAME/MY_PROJECT_NAME.git
   ```

   Alternatively, if you downloaded a ZIP file of your project, extract it.

   ```
   $ unzip MY_PROJECT_NAME.zip
   ```

2. Create a new OpenShift project.

   ```
   $ oc new-project MY_PROJECT_NAME
   ```

3. Navigate to the root directory of your application.

4. Deploy the PostgreSQL database to OpenShift. Ensure that you use the following values for user name, password, and database name when creating your database application. The example application is pre-configured to use these values. Using different values prevents your application from integrating with the database.

   ```
   $ oc new-app -e POSTGRESQL_USER=luke -ePOSTGRESQL_PASSWORD=secret -
   ePOSTGRESQL_DATABASE=my_data registry.access.redhat.com/rhscl/postgresql-10-rhel7
   --name=my-database
   ```

5. Check the status of your database and ensure the pod is running.

   ```
   $ oc get pods -w
   my-database-1-aaaaa   1/1       Running   0         45s
   my-database-1-deploy   0/1       Completed   0         53s
   ```

   The **my-database-1-aaaaa** pod should have a status of **Running** and should be indicated as ready once it is fully deployed and started. Your specific pod name will vary. The number in the middle will increase with each new build. The letters at the end are generated when the pod is created.

6. Use **npm** to start the deployment to OpenShift.

   ```
   $ npm install && npm run openshift
   ```

   These commands install any missing module dependencies, then using the Nodeshift module, deploy the example application on OpenShift.

7. Check the status of your application and ensure your pod is running.

   ```
   $ oc get pods -w
   NAME                          READY     STATUS      RESTARTS   AGE
   MY_APP_NAME-1-aaaaa       1/1       Running     0          58s
   MY_APP_NAME-s2i-1-build   0/1       Completed   0          2m
   ```

   Your **MY_APP_NAME-1-aaaaa** pod should have a status of **Running** and should be indicated as ready once it is fully deployed and started.

8. After your example application is deployed and started, determine its route.

   Example Route Information

   ```
   $ oc get routes
   NAME         HOST/PORT                                    PATH      SERVICES        PORT
   TERMINATION
   MY_APP_NAME   MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME
   MY_APP_NAME   8080
   ```

   The route information of a pod gives you the base URL which you use to access it. In the example above, you would use **http://MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME** as the base URL to access the application.

4.3.4. Deploying the Relational Database Backend example application to OpenShift Container Platform

The process of creating and deploying example applications to OpenShift Container Platform is similar to OpenShift Online:

Prerequisites

- The example application created using developers.redhat.com/launch.

Procedure

- Follow the instructions in Section 4.3.2, "Deploying the Relational Database Backend example application to OpenShift Online", only use the URL and user credentials from the OpenShift Container Platform Web Console.

4.3.5. Interacting with the Relational Database Backend API on Node.js

When you have finished creating your example application, you can interact with it the following way:

Prerequisites

- Your application running
- The **curl** binary or a web browser

Procedure

1. Obtain the URL of your application by executing the following command:

   ```
   $ oc get route MY_APP_NAME
   ```

   ```
   NAME          HOST/PORT                              PATH    SERVICES        PORT
   TERMINATION
   MY_APP_NAME       MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME
   MY_APP_NAME       8080
   ```

2. To access the web interface of the database application, navigate to the *application URL* in your browser:

   ```
   http://MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME
   ```

   Alternatively, you can make requests directly on the **api/fruits**/* endpoint using **curl**:

   #### List all entries in the database:

   ```
   $ curl http://MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME/api/fruits
   ```

   ```
   [ {
     "id" : 1,
     "name" : "Apple",
     "stock" : 10
   }, {
   ```

```
  "id" : 2,
  "name" : "Orange",
  "stock" : 10
}, {
  "id" : 3,
  "name" : "Pear",
  "stock" : 10
} ]
```

Retrieve an entry with a specific ID

```
$ curl http://MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME/api/fruits/3
```

```
{
  "id" : 3,
  "name" : "Pear",
  "stock" : 10
}
```

Create a new entry:

```
$ curl -H "Content-Type: application/json" -X POST -d '{"name":"Peach","stock":1}'
http://MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME/api/fruits
```

```
{
  "id" : 4,
  "name" : "Peach",
  "stock" : 1
}
```

Update an Entry

```
$ curl -H "Content-Type: application/json" -X PUT -d '{"name":"Apple","stock":100}'
http://MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME/api/fruits/1
```

```
{
  "id" : 1,
  "name" : "Apple",
  "stock" : 100
}
```

Delete an Entry:

```
$ curl -X DELETE http://MY_APP_NAME-
MY_PROJECT_NAME.OPENSHIFT_HOSTNAME/api/fruits/1
```

Troubleshooting

- If you receive an HTTP Error code **503** as a response after executing these commands, it means that the application is not ready yet.

4.3.6. Relational database resources

More background and related information on running relational databases in OpenShift, CRUD, HTTP API and REST can be found here:

- HTTP Verbs
- Architectural Styles and the Design of Network-based Software Architectures - Representational State Transfer (REST)
- The never ending REST API design debase
- REST APIs must be Hypertext driven
- Richardson Maturity Model
- Express Web Framework
- Relational Database Backend for Spring Boot
- Relational Database Backend for Eclipse Vert.x
- Relational Database Backend for Thorntail

4.4. HEALTH CHECK EXAMPLE FOR NODE.JS

IMPORTANT

The following example is not meant to be run in a production environment.

Example proficiency level: **Foundational**.

When you deploy an application, it is important to know if it is available and if it can start handling incoming requests. Implementing the *health check* pattern allows you to monitor the health of an application, which includes if an application is available and whether it is able to service requests.

NOTE

If you are not familiar with the health check terminology, see the Section 4.4.1, "Health check concepts" section first.

The purpose of this use case is to demonstrate the health check pattern through the use of probing. Probing is used to report the liveness and readiness of an application. In this use case, you configure an application which exposes an HTTP **health** endpoint to issue HTTP requests. If the container is alive, according to the liveness probe on the **health** HTTP endpoint, the management platform receives **200** as return code and no further action is required. If the **health** HTTP endpoint does not return a response, for example if the thread is blocked, then the application is not considered alive according to the liveness probe. In that case, the platform kills the pod corresponding to that application and recreates a new pod to restart the application.

This use case also allows you to demonstrate and use a readiness probe. In cases where the application is running but is unable to handle requests, such as when the application returns an HTTP **503** response code during restart, this application is not considered ready according to the readiness probe. If the application is not considered ready by the readiness probe, requests are not routed to that application until it is considered ready according to the readiness probe.

4.4.1. Health check concepts

In order to understand the health check pattern, you need to first understand the following concepts:

Liveness

Liveness defines whether an application is running or not. Sometimes a running application moves into an unresponsive or stopped state and needs to be restarted. Checking for liveness helps determine whether or not an application needs to be restarted.

Readiness

Readiness defines whether a running application can service requests. Sometimes a running application moves into an error or broken state where it can no longer service requests. Checking readiness helps determine whether or not requests should continue to be routed to that application.

Fail-over

Fail-over enables failures in servicing requests to be handled gracefully. If an application fails to service a request, that request and future requests can then *fail-over* or be routed to another application, which is usually a redundant copy of that same application.

Resilience and Stability

Resilience and Stability enable failures in servicing requests to be handled gracefully. If an application fails to service a request due to connection loss, in a resilient system that request can be retried after the connection is re-established.

Probe

A probe is a Kubernetes action that periodically performs diagnostics on a running container.

4.4.2. Deploying the Health Check example application to OpenShift Online

Use one of the following options to execute the Health Check example application on OpenShift Online.

- Use developers.redhat.com/launch
- Use the **oc** CLI client

Although each method uses the same **oc** commands to deploy your application, using developers.redhat.com/launch provides an automated deployment workflow that executes the **oc** commands for you.

4.4.2.1. Deploying the example application using developers.redhat.com/launch

Prerequisites

- An account at OpenShift Online.

Procedure

1. Navigate to the developers.redhat.com/launch URL in a browser and log in.
2. Follow on-screen instructions to create and launch your example application in Node.js.

4.4.2.2. Authenticating the oc CLI client

To work with example applications on OpenShift Online using the **oc** command-line client, you must authenticate the client using the token provided by the OpenShift Online web interface.

Prerequisites

- An account at OpenShift Online.

Procedure

1. Navigate to the OpenShift Online URL in a browser.
2. Click on the question mark icon in the top right-hand corner of the Web console, next to your user name.
3. Select *Command Line Tools* in the drop-down menu.
4. Find the text box that contains the **oc login …** command with the hidden token, and click the button next to it to copy its content to your clipboard.
5. Paste the command into a terminal application. The command uses your authentication token to authenticate your **oc** CLI client with your OpenShift Online account.

   ```
   $ oc login OPENSHIFT_URL --token=MYTOKEN
   ```

4.4.2.3. Deploying the Health Check example application using the oc CLI client

Prerequisites

- The example application created using developers.redhat.com/launch. For more information, see Section 4.4.2.1, "Deploying the example application using developers.redhat.com/launch" .
- The **oc** client authenticated. For more information, see Section 4.4.2.2, "Authenticating the **oc** CLI client".

Procedure

1. Clone your project from GitHub.

   ```
   $ git clone git@github.com:USERNAME/MY_PROJECT_NAME.git
   ```

   Alternatively, if you downloaded a ZIP file of your project, extract it.

   ```
   $ unzip MY_PROJECT_NAME.zip
   ```

2. Create a new OpenShift project.

   ```
   $ oc new-project MY_PROJECT_NAME
   ```

3. Navigate to the root directory of your application.
4. Use **npm** to start the deployment to OpenShift.

   ```
   $ npm install && npm run openshift
   ```

   These commands install any missing module dependencies, then using the Nodeshift module, deploy the example application on OpenShift.

5. Check the status of your application and ensure your pod is running.

```
$ oc get pods -w
NAME                             READY     STATUS      RESTARTS   AGE
MY_APP_NAME-1-aaaaa               1/1       Running     0          58s
MY_APP_NAME-s2i-1-build           0/1       Completed   0          2m
```

   The **MY_APP_NAME-1-aaaaa** pod should have a status of **Running** once its fully deployed and started. You should also wait for your pod to be ready before proceeding, which is shown in the **READY** column. For example, **MY_APP_NAME-1-aaaaa** is ready when the **READY** column is **1/1**. Your specific pod name will vary. The number in the middle will increase with each new build. The letters at the end are generated when the pod is created.

6. After your example application is deployed and started, determine its route.

   Example Route Information

```
$ oc get routes
NAME                 HOST/PORT                                                    PATH      SERVICES
PORT      TERMINATION
MY_APP_NAME         MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME
MY_APP_NAME      8080
```

   The route information of a pod gives you the base URL which you use to access it. In the example above, you would use **http://MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME** as the base URL to access the application.

4.4.3. Deploying the Health Check example application to Minishift or CDK

Use one of the following options to execute the Health Check example application locally on Minishift or CDK:

- Using Fabric8 Launcher
- Using the **oc** CLI client

Although each method uses the same **oc** commands to deploy your application, using Fabric8 Launcher provides an automated deployment workflow that executes the **oc** commands for you.

4.4.3.1. Getting the Fabric8 Launcher tool URL and credentials

You need the Fabric8 Launcher tool URL and user credentials to create and deploy example applications on Minishift or CDK. This information is provided when the Minishift or CDK is started.

Prerequisites

- The Fabric8 Launcher tool installed, configured, and running.

Procedure

1. Navigate to the console where you started Minishift or CDK.

2. Check the console output for the URL and user credentials you can use to access the running Fabric8 Launcher:

Example Console Output from a Minishift or CDK Startup

```
...
-- Removing temporary directory ... OK
-- Server Information ...
   OpenShift server started.
   The server is accessible via web console at:
       https://192.168.42.152:8443

   You are logged in as:
       User:     developer
       Password: developer

   To login as administrator:
       oc login -u system:admin
```

4.4.3.2. Deploying the example application using the Fabric8 Launcher tool

Prerequisites

- The URL of your running Fabric8 Launcher instance and the user credentials of your Minishift or CDK. For more information, see Section 4.4.3.1, "Getting the Fabric8 Launcher tool URL and credentials".

Procedure

1. Navigate to the Fabric8 Launcher URL in a browser.
2. Follow the on-screen instructions to create and launch your example application in Node.js.

4.4.3.3. Authenticating the oc CLI client

To work with example applications on Minishift or CDK using the **oc** command-line client, you must authenticate the client using the token provided by the Minishift or CDK web interface.

Prerequisites

- The URL of your running Fabric8 Launcher instance and the user credentials of your Minishift or CDK. For more information, see Section 4.4.3.1, "Getting the Fabric8 Launcher tool URL and credentials".

Procedure

1. Navigate to the Minishift or CDK URL in a browser.
2. Click on the question mark icon in the top right-hand corner of the Web console, next to your user name.
3. Select *Command Line Tools* in the drop-down menu.
4. Find the text box that contains the **oc login …** command with the hidden token, and click the button next to it to copy its content to your clipboard.

5. Paste the command into a terminal application. The command uses your authentication token to authenticate your **oc** CLI client with your Minishift or CDK account.

```
$ oc login OPENSHIFT_URL --token=MYTOKEN
```

4.4.3.4. Deploying the Health Check example application using the **oc** CLI client

Prerequisites

- The example application created using Fabric8 Launcher tool on a Minishift or CDK. For more information, see Section 4.4.3.2, "Deploying the example application using the Fabric8 Launcher tool".
- Your Fabric8 Launcher tool URL.
- The **oc** client authenticated. For more information, see Section 4.4.3.3, "Authenticating the **oc** CLI client".

Procedure

1. Clone your project from GitHub.

```
$ git clone git@github.com:USERNAME/MY_PROJECT_NAME.git
```

Alternatively, if you downloaded a ZIP file of your project, extract it.

```
$ unzip MY_PROJECT_NAME.zip
```

2. Create a new OpenShift project.

```
$ oc new-project MY_PROJECT_NAME
```

3. Navigate to the root directory of your application.

4. Use **npm** to start the deployment to OpenShift.

```
$ npm install && npm run openshift
```

These commands install any missing module dependencies, then using the Nodeshift module, deploy the example application on OpenShift.

5. Check the status of your application and ensure your pod is running.

```
$ oc get pods -w
NAME                             READY     STATUS      RESTARTS   AGE
MY_APP_NAME-1-aaaaa               1/1       Running     0          58s
MY_APP_NAME-s2i-1-build           0/1       Completed   0          2m
```

The **MY_APP_NAME-1-aaaaa** pod should have a status of **Running** once its fully deployed and started. You should also wait for your pod to be ready before proceeding, which is shown in the **READY** column. For example, **MY_APP_NAME-1-aaaaa** is ready when the **READY** column is **1/1**. Your specific pod name will vary. The number in the middle will increase with each new build. The letters at the end are generated when the pod is created.

6. After your example application is deployed and started, determine its route.

   Example Route Information

   ```
   $ oc get routes
   NAME            HOST/PORT                                    PATH     SERVICES
   PORT     TERMINATION
   MY_APP_NAME       MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME
   MY_APP_NAME     8080
   ```

   The route information of a pod gives you the base URL which you use to access it. In the example above, you would use **http://MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME** as the base URL to access the application.

4.4.4. Deploying the Health Check example application to OpenShift Container Platform

The process of creating and deploying example applications to OpenShift Container Platform is similar to OpenShift Online:

Prerequisites

- The example application created using developers.redhat.com/launch.

Procedure

- Follow the instructions in Section 4.4.2, “Deploying the Health Check example application to OpenShift Online”, only use the URL and user credentials from the OpenShift Container Platform Web Console.

4.4.5. Interacting with the unmodified Health Check example application

After you deploy the example application, you will have the **MY_APP_NAME** service running. The **MY_APP_NAME** service exposes the following REST endpoints:

/api/greeting

Returns a JSON containing greeting of **name** parameter (or World as default value).

/api/stop

Forces the service to become unresponsive as means to simulate a failure.

The following steps demonstrate how to verify the service availability and simulate a failure. This failure of an available service causes the OpenShift self-healing capabilities to be trigger on the service.

Alternatively, you can use the web interface to perform these steps.

1. Use **curl** to execute a **GET** request against the **MY_APP_NAME** service. You can also use a browser to do this.

   ```
   $ curl http://MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME/api/greeting
   ```

   ```
   {"content":"Hello, World!"}
   ```

2. Invoke the **/api/stop** endpoint and verify the availability of the **/api/greeting** endpoint shortly after that.
   Invoking the **/api/stop** endpoint simulates an internal service failure and triggers the OpenShift self-healing capabilities. When invoking **/api/greeting** after simulating the failure, the service should return a HTTP status **503**.

   ```
   $ curl http://MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME/api/stop
   ```

   ```
   Stopping HTTP server, Bye bye world !
   ```

   (followed by)

   ```
   $ curl http://MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME/api/greeting
   ```

   ```
   Not online
   ```

3. Use **oc get pods -w** to continuously watch the self-healing capabilities in action.
   While invoking the service failure, you can watch the self-healing capabilities in action on OpenShift console, or with the **oc** client tools. You should see the number of pods in the **READY** state move to zero (**0/1**) and after a short period (less than one minute) move back up to one (**1/1**). In addition to that, the **RESTARTS** count increases every time you you invoke the service failure.

   ```
   $ oc get pods -w
   NAME                     READY     STATUS    RESTARTS   AGE
   MY_APP_NAME-1-26iy7   0/1       Running   5          18m
   MY_APP_NAME-1-26iy7   1/1       Running   5         19m
   ```

4. Optional: Use the web interface to invoke the service.
   Alternatively to the interaction using the terminal window, you can use the web interface provided by the service to invoke the different methods and watch the service move through the life cycle phases.

   ```
   http://MY_APP_NAME-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME
   ```

5. Optional: Use the web console to view the log output generated by the application at each stage of the self-healing process.
   1. Navigate to your project.
   2. On the sidebar, click on *Monitoring*.
   3. In the upper right-hand corner of the screen, click on *Events* to display the log messages.
   4. Optional: Click *View Details* to display a detailed view of the Event log.

   The health check application generates the following messages:

Message	Status

Message	Status
Unhealthy	Readiness probe failed. This message is expected and indicates that the simulated failure of the **/api/greeting** endpoint has been detected and the self-healing process starts.
Killing	The unavailable Docker container running the service is being killed before being re-created.
Pulling	Downloading the latest version of docker image to re-create the container.
Pulled	Docker image downloaded successfully.
Created	Docker container has been successfully created
Started	Docker container is ready to handle requests

4.4.6. Health check resources

More background and related information on health checking can be found here:

- Application Health in OpenShift
- Kubernetes Liveness and Readiness Probes
- Health Check for Spring Boot
- Health Check for Eclipse Vert.x
- Health Check for Thorntail

4.5. CIRCUIT BREAKER EXAMPLE FOR NODE.JS

IMPORTANT

The following example is not meant to be run in a production environment.

Limitation: Run this example application on a Minishift or CDK. You can also use a manual workflow to deploy this example to OpenShift Online Pro and OpenShift Container Platform. This example is not currently available on OpenShift Online Starter.

Example proficiency level: **Foundational**.

The *Circuit Breaker* example demonstrates a generic pattern for reporting the failure of a service and then limiting access to the failed service until it becomes available to handle requests. This helps prevent cascading failure in other services that depend on the failed services for functionality.

This example shows you how to implement a Circuit Breaker and Fallback pattern in your services.

4.5.1. The circuit breaker design pattern

The Circuit Breaker is a pattern intended to:

- Reduce the impact of network failure and high latency on service architectures where services synchronously invoke other services.
  If one of the services:
  - becomes unavailable due to network failure, or
  - incurs unusually high latency values due to overwhelming traffic,

  other services attempting to call its endpoint may end up exhausting critical resources in an attempt to reach it, rendering themselves unusable.
- Prevent the condition also known as cascading failure, which can render the entire microservice architecture unusable.
- Act as a proxy between a protected function and a remote function, which monitors for failures.
- Trip once the failures reach a certain threshold, and all further calls to the circuit breaker return an error or a predefined fallback response, without the protected call being made at all.

The Circuit Breaker usually also contain an error reporting mechanism that notifies you when the Circuit Breaker trips.

Circuit breaker implementation

- With the Circuit Breaker pattern implemented, a service client invokes a remote service endpoint via a proxy at regular intervals.
- If the calls to the remote service endpoint fail repeatedly and consistently, the Circuit Breaker trips, making all calls to the service fail immediately over a set timeout period and returns a predefined fallback response.
- When the timeout period expires, a limited number of test calls are allowed to pass through to the remote service to determine whether it has healed, or remains unavailable.
  - If the test calls fail, the Circuit Breaker keeps the service unavailable and keeps returning the fallback responses to incoming calls.
  - If the test calls succeed, the Circuit Breaker closes, fully enabling traffic to reach the remote service again.

4.5.2. Circuit Breaker design tradeoffs

Table 4.4. Design Tradeoffs

Pros	Cons

Pros	Cons
• Enables a service to handle the failure of other services it invokes.	• Optimizing the timeout values can be challenging ◦ Larger-than-necessary timeout values may generate excessive latency. ◦ Smaller-than-necessary timeout values may introduce false positives.

4.5.3. Deploying the Circuit Breaker example application to OpenShift Online

Use one of the following options to execute the Circuit Breaker example application on OpenShift Online.

- Use developers.redhat.com/launch
- Use the **oc** CLI client

Although each method uses the same **oc** commands to deploy your application, using developers.redhat.com/launch provides an automated deployment workflow that executes the **oc** commands for you.

4.5.3.1. Deploying the example application using developers.redhat.com/launch

Prerequisites

- An account at OpenShift Online.

Procedure

1. Navigate to the developers.redhat.com/launch URL in a browser and log in.
2. Follow on-screen instructions to create and launch your example application in Node.js.

4.5.3.2. Authenticating the oc CLI client

To work with example applications on OpenShift Online using the **oc** command-line client, you must authenticate the client using the token provided by the OpenShift Online web interface.

Prerequisites

- An account at OpenShift Online.

Procedure

1. Navigate to the OpenShift Online URL in a browser.
2. Click on the question mark icon in the top right-hand corner of the Web console, next to your user name.
3. Select *Command Line Tools* in the drop-down menu.

4. Find the text box that contains the **oc login …** command with the hidden token, and click the button next to it to copy its content to your clipboard.

5. Paste the command into a terminal application. The command uses your authentication token to authenticate your **oc** CLI client with your OpenShift Online account.

```
$ oc login OPENSHIFT_URL --token=MYTOKEN
```

4.5.3.3. Deploying the Circuit Breaker example application using the oc CLI client

Prerequisites

- The example application created using developers.redhat.com/launch. For more information, see Section 4.5.3.1, "Deploying the example application using developers.redhat.com/launch" .
- The **oc** client authenticated. For more information, see Section 4.5.3.2, "Authenticating the **oc** CLI client".

Procedure

1. Clone your project from GitHub.

```
$ git clone git@github.com:USERNAME/MY_PROJECT_NAME.git
```

   Alternatively, if you downloaded a ZIP file of your project, extract it.

```
$ unzip MY_PROJECT_NAME.zip
```

2. Create a new OpenShift project.

```
$ oc new-project MY_PROJECT_NAME
```

3. Navigate to the root directory of your application.

4. Use the provided **start-openshift.sh** script to start the deployment to OpenShift.

```
$ chmod +x start-openshift.sh
$ ./start-openshift.sh
```

   These commands use the Nodeshift **npm** module to install your dependencies, launch the S2I build process on OpenShift, and start the services.

5. Check the status of your application and ensure your pod is running.

```
$ oc get pods -w
NAME                           READY     STATUS      RESTARTS   AGE
MY_APP_NAME-greeting-1-aaaaa     1/1       Running   0          17s
MY_APP_NAME-greeting-1-deploy   0/1       Completed 0          22s
MY_APP_NAME-name-1-aaaaa         1/1       Running   0          14s
MY_APP_NAME-name-1-deploy       0/1       Completed 0          28s
```

   Both the **MY_APP_NAME-greeting-1-aaaaa** and **MY_APP_NAME-name-1-aaaaa** pods should have a status of **Running** once they are fully deployed and started. You should also wait for your pods to be ready before proceeding, which is shown in the **READY** column. For example, **MY_APP_NAME-greeting-1-aaaaa** is ready when the **READY** column is **1/1**. Your specific pod names will vary. The number in the middle will increase with each new build. The letters at the end are generated when the pod is created.

6. After your example application is deployed and started, determine its route.

   Example Route Information

```
$ oc get routes
NAME                HOST/PORT                                          PATH      SERVICES
PORT      TERMINATION
MY_APP_NAME-greeting   MY_APP_NAME-greeting-
MY_PROJECT_NAME.OPENSHIFT_HOSTNAME         MY_APP_NAME-greeting   8080
None
MY_APP_NAME-name       MY_APP_NAME-name-
MY_PROJECT_NAME.OPENSHIFT_HOSTNAME         MY_APP_NAME-name       8080
None
```

   The route information of a pod gives you the base URL which you use to access it. In the example above, you would use **http://MY_APP_NAME-greeting-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME** as the base URL to access the application.

4.5.4. Deploying the Circuit Breaker example application to Minishift or CDK

Use one of the following options to execute the Circuit Breaker example application locally on Minishift or CDK:

- Using Fabric8 Launcher
- Using the **oc** CLI client

Although each method uses the same **oc** commands to deploy your application, using Fabric8 Launcher provides an automated deployment workflow that executes the **oc** commands for you.

4.5.4.1. Getting the Fabric8 Launcher tool URL and credentials

You need the Fabric8 Launcher tool URL and user credentials to create and deploy example applications on Minishift or CDK. This information is provided when the Minishift or CDK is started.

Prerequisites

- The Fabric8 Launcher tool installed, configured, and running.

Procedure

1. Navigate to the console where you started Minishift or CDK.
2. Check the console output for the URL and user credentials you can use to access the running Fabric8 Launcher:

   Example Console Output from a Minishift or CDK Startup

```
...
-- Removing temporary directory ... OK
-- Server Information ...
  OpenShift server started.
  The server is accessible via web console at:
     https://192.168.42.152:8443

  You are logged in as:
     User:     developer
     Password: developer

  To login as administrator:
     oc login -u system:admin
```

4.5.4.2. Deploying the example application using the Fabric8 Launcher tool

Prerequisites

- The URL of your running Fabric8 Launcher instance and the user credentials of your Minishift or CDK. For more information, see Section 4.5.4.1, "Getting the Fabric8 Launcher tool URL and credentials".

Procedure

1. Navigate to the Fabric8 Launcher URL in a browser.
2. Follow the on-screen instructions to create and launch your example application in Node.js.

4.5.4.3. Authenticating the oc CLI client

To work with example applications on Minishift or CDK using the **oc** command-line client, you must authenticate the client using the token provided by the Minishift or CDK web interface.

Prerequisites

- The URL of your running Fabric8 Launcher instance and the user credentials of your Minishift or CDK. For more information, see Section 4.5.4.1, "Getting the Fabric8 Launcher tool URL and credentials".

Procedure

1. Navigate to the Minishift or CDK URL in a browser.
2. Click on the question mark icon in the top right-hand corner of the Web console, next to your user name.
3. Select *Command Line Tools* in the drop-down menu.
4. Find the text box that contains the **oc login …** command with the hidden token, and click the button next to it to copy its content to your clipboard.
5. Paste the command into a terminal application. The command uses your authentication token to authenticate your **oc** CLI client with your Minishift or CDK account.

```
$ oc login OPENSHIFT_URL --token=MYTOKEN
```

4.5.4.4. Deploying the Circuit Breaker example application using the oc CLI client

Prerequisites

- The example application created using Fabric8 Launcher tool on a Minishift or CDK. For more information, see Section 4.5.4.2, "Deploying the example application using the Fabric8 Launcher tool".
- Your Fabric8 Launcher tool URL.
- The **oc** client authenticated. For more information, see Section 4.5.4.3, "Authenticating the **oc** CLI client".

Procedure

1. Clone your project from GitHub.

   ```
   $ git clone git@github.com:USERNAME/MY_PROJECT_NAME.git
   ```

   Alternatively, if you downloaded a ZIP file of your project, extract it.

   ```
   $ unzip MY_PROJECT_NAME.zip
   ```

2. Create a new OpenShift project.

   ```
   $ oc new-project MY_PROJECT_NAME
   ```

3. Navigate to the root directory of your application.
4. Use the provided **start-openshift.sh** script to start the deployment to OpenShift.

   ```
   $ chmod +x start-openshift.sh
   $ ./start-openshift.sh
   ```

   These commands use the Nodeshift **npm** module to install your dependencies, launch the S2I build process on OpenShift, and start the services.

5. Check the status of your application and ensure your pod is running.

   ```
   $ oc get pods -w
   NAME                           READY     STATUS      RESTARTS   AGE
   MY_APP_NAME-greeting-1-aaaaa     1/1       Running   0          17s
   MY_APP_NAME-greeting-1-deploy   0/1       Completed 0          22s
   MY_APP_NAME-name-1-aaaaa         1/1       Running   0          14s
   MY_APP_NAME-name-1-deploy       0/1       Completed 0          28s
   ```

   Both the **MY_APP_NAME-greeting-1-aaaaa** and **MY_APP_NAME-name-1-aaaaa** pods should have a status of **Running** once they are fully deployed and started. You should also wait for your pods to be ready before proceeding, which is shown in the **READY** column. For example, **MY_APP_NAME-greeting-1-aaaaa** is ready when the **READY** column is **1/1**. Your specific pod names will vary. The number in the middle will increase with each new build. The letters at the end are generated when the pod is created.

6. After your example application is deployed and started, determine its route.

Example Route Information

```
$ oc get routes
NAME              HOST/PORT                                   PATH    SERVICES
PORT     TERMINATION
MY_APP_NAME-greeting   MY_APP_NAME-greeting-
MY_PROJECT_NAME.OPENSHIFT_HOSTNAME          MY_APP_NAME-greeting   8080
None
MY_APP_NAME-name       MY_APP_NAME-name-
MY_PROJECT_NAME.OPENSHIFT_HOSTNAME          MY_APP_NAME-name       8080
None
```

The route information of a pod gives you the base URL which you use to access it. In the example above, you would use **http://MY_APP_NAME-greeting-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME** as the base URL to access the application.

4.5.5. Deploying the Circuit Breaker example application to OpenShift Container Platform

The process of creating and deploying example applications to OpenShift Container Platform is similar to OpenShift Online:

Prerequisites

- The example application created using developers.redhat.com/launch.

Procedure

- Follow the instructions in Section 4.5.3, "Deploying the Circuit Breaker example application to OpenShift Online", only use the URL and user credentials from the OpenShift Container Platform Web Console.

4.5.6. Interacting with the unmodified Node.js Circuit Breaker example application

After you have the Node.js example application deployed, you have the following services running:

MY_APP_NAME-name

Exposes the following endpoints:

- the **/api/name** endpoint, which returns a name when this service is working, and an error when this service is set up to demonstrate failure.
- the **/api/state** endpoint, which controls the behavior of the **/api/name** endpoint and determines whether the service works correctly or demonstrates failure.

MY_APP_NAME-greeting

Exposes the following endpoints:

- the **/api/greeting** endpoint that you can call to get a personalized greeting response.
  When you call the **/api/greeting** endpoint, it issues a call against the **/api/name** endpoint of the **MY_APP_NAME-name** service as part of processing your request. The call made against the **/api/name** endpoint is protected by the Circuit Breaker.

If the remote endpoint is available, the **name** service responds with an HTTP code **200** (**OK**) and you receive the following greeting from the **/api/greeting** endpoint:

```
{"content":"Hello, World!"}
```

If the remote endpoint is unavailable, the **name** service responds with an HTTP code **500** (**Internal server error**) and you receive a predefined fallback response from the **/api/greeting** endpoint:

```
{"content":"Hello, Fallback!"}
```

- the **/api/cb-state** endpoint, which returns the state of the Circuit Breaker. The state can be:
  - *open* : the circuit breaker is preventing requests from reaching the failed service,
  - *closed*: the circuit breaker is allowing requests to reach the service.

The following steps demonstrate how to verify the availability of the service, simulate a failure and receive a fallback response.

1. Use **curl** to execute a **GET** request against the **MY_APP_NAME-greeting** service. You can also use the **Invoke** button in the web interface to do this.

   ```
   $ curl http://MY_APP_NAME-greeting-
   MY_PROJECT_NAME.LOCAL_OPENSHIFT_HOSTNAME/api/greeting
   {"content":"Hello, World!"}
   ```

2. To simulate the failure of the **MY_APP_NAME-name** service you can:
   - use the **Toggle** button in the web interface.
   - scale the number of replicas of the pod running the **MY_APP_NAME-name** service down to 0.
   - execute an HTTP **PUT** request against the **/api/state** endpoint of the **MY_APP_NAME-name** service to set its state to **fail**.

     ```
     $ curl -X PUT -H "Content-Type: application/json" -d '{"state": "fail"}'
     http://MY_APP_NAME-name-
     MY_PROJECT_NAME.LOCAL_OPENSHIFT_HOSTNAME/api/state
     ```

3. Invoke the **/api/greeting** endpoint. When several requests on the **/api/name** endpoint fail:
   a. the Circuit Breaker opens,
   b. the state indicator in the web interface changes from **CLOSED** to **OPEN**,
   c. the Circuit Breaker issues a fallback response when you invoke the **/api/greeting** endpoint:

     ```
     $ curl http://MY_APP_NAME-greeting-
     MY_PROJECT_NAME.LOCAL_OPENSHIFT_HOSTNAME/api/greeting
     {"content":"Hello, Fallback!"}
     ```

4. Restore the name **MY_APP_NAME-name** service to availability. To do this you can:

- use the **Toggle** button in the web interface.
- scale the number of replicas of the pod running the **MY_APP_NAME-name** service back up to 1.
- execute an HTTP **PUT** request against the **/api/state** endpoint of the **MY_APP_NAME-name** service to set its state back to **ok**.

```
$ curl -X PUT -H "Content-Type: application/json" -d '{"state": "ok"}'
http://MY_APP_NAME-name-
MY_PROJECT_NAME.LOCAL_OPENSHIFT_HOSTNAME/api/state
```

5. Invoke the **/api/greeting** endpoint again. When several requests on the **/api/name** endpoint succeed:

   a. the Circuit Breaker closes,

   b. the state indicator in the web interface changes from **OPEN** to **CLOSED**,

   c. the Circuit Breaker issues a returns the **Hello World!** greeting when you invoke the **/api/greeting** endpoint:

```
$ curl http://MY_APP_NAME-greeting-
MY_PROJECT_NAME.LOCAL_OPENSHIFT_HOSTNAME/api/greeting
{"content":"Hello, World!"}
```

4.5.7. Circuit breaker resources

Follow the links below for more background information on the design principles behind the Circuit Breaker pattern

- microservices.io: Microservice Patterns: Circuit Breaker
- Martin Fowler: CircuitBreaker
- Circuit Breaker for Spring Boot
- Circuit Breaker for Eclipse Vert.x
- Circuit Breaker for Thorntail

4.6. SECURED EXAMPLE APPLICATION FOR NODE.JS

IMPORTANT

The following example is not meant to be run in a production environment.

Limitation: Run this example application on a Minishift or CDK. You can also use a manual workflow to deploy this example to OpenShift Online Pro and OpenShift Container Platform. This example is not currently available on OpenShift Online Starter.

NOTE

The Secured example application in Node.js requires Red Hat SSO 7.3. Since Red Hat SSO 7.3 is not supported on IBM Z, the Secured example is not available for IBM Z.

Example proficiency level: **Advanced**.

The Secured example application secures a REST endpoint using Red Hat SSO . (This example expands on the REST API Level 0 example).

Red Hat SSO:

- Implements the Open ID Connect protocol which is an extension of the OAuth 2.0 specification.
- Issues access tokens to provide clients with various access rights to secured resources.

Securing an application with SSO enables you to add security to your applications while centralizing the security configuration.

IMPORTANT

This example comes with Red Hat SSO pre-configured for demonstration purposes, it does not explain its principles, usage, or configuration. Before using this example, ensure that you are familiar with the basic concepts related to Red Hat SSO .

4.6.1. The Secured project structure

The SSO example contains:

- the sources for the Greeting service, which is the one which we are going to to secure
- a template file (**service.sso.yaml**) to deploy the SSO server
- the Keycloak adapter configuration to secure the service

4.6.2. Red Hat SSO deployment configuration

The **service.sso.yaml** file in this example contains all OpenShift configuration items to deploy a pre-configured Red Hat SSO server. The SSO server configuration has been simplified for the sake of this exercise and does provide an out-of-the-box configuration, with pre-configured users and security settings. The **service.sso.yaml** file also contains very long lines, and some text editors, such as gedit, may have issues reading this file.

WARNING

It is not recommended to use this SSO configuration in production. Specifically, the simplifications made to the example security configuration impact the ability to use it in a production environment.

Table 4.5. SSO Example Simplifications

Change	Reason	Recommendation
The default configuration includes both public and **private keys in the yaml configuration files**.	We did this because the end user can deploy Red Hat SSO module and have it in a usable state without needing to know the internals or how to configure Red Hat SSO.	In production, do not store private keys under source control. They should be added by the server administrator.
The configured **clients accept any callback url**.	To avoid having a custom configuration for each runtime, we avoid the callback verification that is required by the OAuth2 specification.	An application-specific callback URL should be provided with a valid domain name.
Clients do not require SSL/TLS and the secured applications are not exposed over HTTPS.	The examples are simplified by not requiring certificates generated for each runtime.	In production a secure application should use HTTPS rather than plain HTTP.
The token timeout has been increased to 10 minutes from the default of 1 minute.	Provides a better user experience when working with the command line examples	From a security perspective, the window an attacker would have to guess the access token is extended. It is recommended to keep this window short as it makes it much harder for a potential attacker to guess the current token.

4.6.3. Red Hat SSO realm model

The **master** realm is used to secure this example. There are two pre-configured application client definitions that provide a model for command line clients and the secured REST endpoint.

There are also two pre-configured users in the Red Hat SSO **master** realm that can be used to validate various authentication and authorization outcomes: **admin** and **alice**.

4.6.3.1. Red Hat SSO users

The realm model for the secured examples includes two users:

admin

The **admin** user has a password of **admin** and is the realm administrator. This user has full access to the Red Hat SSO administration console, but none of the role mappings that are required to access the secured endpoints. You can use this user to illustrate the behavior of an authenticated, but unauthorized user.

alice

The **alice** user has a password of **password** and is the canonical application user. This user will demonstrate successful authenticated and authorized access to the secured endpoints. An example representation of the role mappings is provided in this decoded JWT bearer token:

```
{
  "jti": "0073cfaa-7ed6-4326-ac07-c108d34b4f82",
  "exp": 1510162193,
  "nbf": 0,
  "iat": 1510161593,
  "iss": "https://secure-sso-sso.LOCAL_OPENSHIFT_HOSTNAME/auth/realms/master", ❶
  "aud": "demoapp",
  "sub": "c0175ccb-0892-4b31-829f-dda873815fe8",
  "typ": "Bearer",
  "azp": "demoapp",
  "nonce": "90ff5d1a-ba44-45ae-a413-50b08bf4a242",
  "auth_time": 1510161591,
  "session_state": "98efb95a-b355-43d1-996b-0abcb1304352",
  "acr": "1",
  "client_session": "5962112c-2b19-461e-8aac-84ab512d2a01",
  "allowed-origins": [
    "*"
  ],
  "realm_access": {
    "roles": [ ❷
      "example-admin"
    ]
  },
  "resource_access": { ❸
    "secured-example-endpoint": {
      "roles": [
        "example-admin" ❹
      ]
    },
    "account": {
      "roles": [
        "manage-account",
        "view-profile"
      ]
    }
  },
  "name": "Alice InChains",
  "preferred_username": "alice", ❺
  "given_name": "Alice",
  "family_name": "InChains",
  "email": "alice@keycloak.org"
}
```

❶ The **iss** field corresponds to the Red Hat SSO realm instance URL that issues the token. This must be configured in the secured endpoint deployments in order for the token to be verified.

❷ The **roles** object provides the roles that have been granted to the user at the global realm level. In this case **alice** has been granted the **example-admin** role. We will see that the secured endpoint will look to the realm level for authorized roles.

❸ The **resource_access** object contains resource specific role grants. Under this object you will find an object for each of the secured endpoints.

❹ The **resource_access.secured-example-endpoint.roles** object contains the roles granted to **alice** for the **secured-example-endpoint** resource.

❺ The **preferred_username** field provides the username that was used to generate the access token.

4.6.3.2. The application clients

The OAuth 2.0 specification allows you to define a role for application clients that access secured resources on behalf of resource owners. The **master** realm has the following application clients defined:

demoapp

This is a **confidential** type client with a client secret that is used to obtain an access token. The token contains grants for the **alice** user which enable **alice** to access the Thorntail, Eclipse Vert.x, Node.js and Spring Boot based REST example application deployments.

secured-example-endpoint

The **secured-example-endpoint** is a bearer-only type of client that requires a **example-admin** role for accessing the associated resources, specifically the Greeting service.

4.6.4. Node.js SSO adapter configuration

The SSO adapter is the *client side*, or client to the SSO server, component that enforces security on the web resources. In this specific case, it is the Greeting service.

Enacting Security Example Node.js Code

```
const express = require('express');
const Keycloak = require('keycloak-connect'); 1
const kc = new Keycloak({}); 2

const app = express();

app.use(kc.middleware()); 3

app.use('/api/greeting', kc.protect('example-admin'), callback); 4
```

1 **npm** module keycloak-connect must be installed and **required**. The keycloak-connect module acts as connect middleware, which provides integration with **express**.

2 Instantiate a new **Keycloak** object and pass in an empty configuration object.

3 Tells **express** to use Keycloak as middleware.

4 Enforces that a user must be authenticated and part of the example-admin role before accessing a resource.

Enacting Security in Keycloak Adapter using **keycloak.json**

```
{
  "realm": "master", 1
  "resource": "secured-example-endpoint", 2
  "realm-public-key": "...", 3
  "auth-server-url": "${env.SSO_AUTH_SERVER_URL}", 4
  "ssl-required": "external",
  "disable-trust-manager": true,
  "bearer-only": true, 5
  "use-resource-role-mappings": true
}
```

1. The security realm to be used.
2. The actual Keycloak *client* configuration.
3. PEM format of the realm public key. You can obtain this from the administration console.
4. The address of the Red Hat SSO server (Interpolation at build time).
5. If enabled the adapter will not attempt to authenticate users, but only verify bearer tokens.

The example Node.js code enables Keycloak and enforces protection of the Greeting service web resource endpoint. The **keycloak.json** configures the security adapter to interact with Red Hat SSO.

Additional resources

- For more information about the Node.js Keycloak adapter, see the Keycloak documentation.

4.6.5. Deploying the Secured example application to Minishift or CDK

4.6.5.1. Getting the Fabric8 Launcher tool URL and credentials

You need the Fabric8 Launcher tool URL and user credentials to create and deploy example applications on Minishift or CDK. This information is provided when the Minishift or CDK is started.

Prerequisites

- The Fabric8 Launcher tool installed, configured, and running.

Procedure

1. Navigate to the console where you started Minishift or CDK.
2. Check the console output for the URL and user credentials you can use to access the running Fabric8 Launcher:

   Example Console Output from a Minishift or CDK Startup

```
...
-- Removing temporary directory ... OK
-- Server Information ...
   OpenShift server started.
   The server is accessible via web console at:
       https://192.168.42.152:8443

   You are logged in as:
       User:     developer
       Password: developer

   To login as administrator:
       oc login -u system:admin
```

4.6.5.2. Creating the Secured example application using Fabric8 Launcher

Prerequisites

- The URL and user credentials of your running Fabric8 Launcher instance. For more information, see Section 4.6.5.1, "Getting the Fabric8 Launcher tool URL and credentials" .

Procedure

- Navigate to the Fabric8 Launcher URL in a browser and log in.
- Follow the on-screen instructions to create your example in Node.js. When asked about which deployment type, select *I will build and run locally.*
- Follow on-screen instructions.
  When done, click the **Download as ZIP file**button and store the file on your hard drive.

4.6.5.3. Authenticating the oc CLI client

To work with example applications on Minishift or CDK using the **oc** command-line client, you must authenticate the client using the token provided by the Minishift or CDK web interface.

Prerequisites

- The URL of your running Fabric8 Launcher instance and the user credentials of your Minishift or CDK. For more information, see Section 4.6.5.1, "Getting the Fabric8 Launcher tool URL and credentials".

Procedure

1. Navigate to the Minishift or CDK URL in a browser.
2. Click on the question mark icon in the top right-hand corner of the Web console, next to your user name.
3. Select *Command Line Tools* in the drop-down menu.
4. Find the text box that contains the **oc login …** command with the hidden token, and click the button next to it to copy its content to your clipboard.
5. Paste the command into a terminal application. The command uses your authentication token to authenticate your **oc** CLI client with your Minishift or CDK account.

```
$ oc login OPENSHIFT_URL --token=MYTOKEN
```

4.6.5.4. Deploying the Secured example application using the oc CLI client

Prerequisites

- The example application created using the Fabric8 Launcher tool on a Minishift or CDK. For more information, see Section 4.6.5.2, "Creating the Secured example application using Fabric8 Launcher".
- Your Fabric8 Launcher URL.
- The **oc** client authenticated. For more information, see Section 4.6.5.3, "Authenticating the **oc** CLI client".

Procedure

1. Clone your project from GitHub.

   ```
   $ git clone git@github.com:USERNAME/MY_PROJECT_NAME.git
   ```

   Alternatively, if you downloaded a ZIP file of your project, extract it.

   ```
   $ unzip MY_PROJECT_NAME.zip
   ```

2. Create a new OpenShift project.

   ```
   $ oc new-project MY_PROJECT_NAME
   ```

3. Navigate to the root directory of your application.

4. Deploy the Red Hat SSO server using the **service.sso.yaml** file from your example ZIP file:

   ```
   $ oc create -f service.sso.yaml
   ```

5. Use **npm** to start the deployment to Minishift or CDK.

   ```
   $ npm install && npm run openshift -- \
         -d SSO_AUTH_SERVER_URL=$(oc get route secure-sso -o jsonpath='{"https://"}
   {.spec.host}{"/auth\n"}')
   ```

   These commands install any missing module dependencies, then using the Nodeshift module, deploy the example application on OpenShift.

4.6.6. Deploying the Secured example application to OpenShift Container Platform

In addition to the Minishift or CDK, you can create and deploy the example on OpenShift Container Platform with only minor differences. The most important difference is that you need to create the example application on Minishift or CDK before you can deploy it with OpenShift Container Platform.

Prerequisites

- The example created using Minishift or CDK.

4.6.6.1. Authenticating the oc CLI client

To work with example applications on OpenShift Container Platform using the **oc** command-line client, you must authenticate the client using the token provided by the OpenShift Container Platform web interface.

Prerequisites

- An account at OpenShift Container Platform.

Procedure

1. Navigate to the OpenShift Container Platform URL in a browser.

2. Click on the question mark icon in the top right-hand corner of the Web console, next to your user name.

3. Select *Command Line Tools* in the drop-down menu.

4. Find the text box that contains the **oc login …** command with the hidden token, and click the button next to it to copy its content to your clipboard.

5. Paste the command into a terminal application. The command uses your authentication token to authenticate your **oc** CLI client with your OpenShift Container Platform account.

   ```
   $ oc login OPENSHIFT_URL --token=MYTOKEN
   ```

4.6.6.2. Deploying the Secured example application using the oc CLI client

Prerequisites

- The example application created using the Fabric8 Launcher tool on a Minishift or CDK.
- The **oc** client authenticated. For more information, see Section 4.6.6.1, “Authenticating the **oc** CLI client”.

Procedure

1. Clone your project from GitHub.

   ```
   $ git clone git@github.com:USERNAME/MY_PROJECT_NAME.git
   ```

   Alternatively, if you downloaded a ZIP file of your project, extract it.

   ```
   $ unzip MY_PROJECT_NAME.zip
   ```

2. Create a new OpenShift project.

   ```
   $ oc new-project MY_PROJECT_NAME
   ```

3. Navigate to the root directory of your application.

4. Deploy the Red Hat SSO server using the **service.sso.yaml** file from your example ZIP file:

   ```
   $ oc create -f service.sso.yaml
   ```

5. Use **npm** to start the deployment to OpenShift Container Platform.

   ```
   $ npm install && npm run openshift -- \
       -d SSO_AUTH_SERVER_URL=$(oc get route secure-sso -o jsonpath='{"https://"}
   {.spec.host}{"/auth\n"}')
   ```

   These commands install any missing module dependencies, then using the Nodeshift module, deploy the example application on OpenShift.

4.6.7. Authenticating to the Secured example application API endpoint

The Secured example application provides a default HTTP endpoint that accepts **GET** requests if the caller is authenticated and authorized. The client first authenticates against the Red Hat SSO server and then performs a **GET** request against the Secured example application using the access token returned by the authentication step.

4.6.7.1. Getting the Secured example application API endpoint

When using a client to interact with the example, you must specify the Secured example application endpoint, which is the *PROJECT_ID* service.

Prerequisites

- The Secured example application deployed and running.
- The **oc** client authenticated.

Procedure

1. In a terminal application, execute the **oc get routes** command.
   A sample output is shown in the following table:

Example 4.1. List of Secured endpoints

Name	Host/Port	Path	Services	Port	Termination
secure-sso	secure-sso-myproject.LOCAL_OPENSHIFT_HOSTNAME		secure-sso	<all>	passthrough
PROJECT_ID	PROJECT_ID-myproject.LOCAL_OPENSHIFT_HOSTNAME		PROJECT_ID	<all>	
sso	sso-myproject.LOCAL_OPENSHIFT_HOSTNAME		sso	<all>	

In the above example, the example endpoint would be **http://PROJECT_ID-myproject.LOCAL_OPENSHIFT_HOSTNAME**. **PROJECT_ID** is based on the name you entered when generating your example using developers.redhat.com/launch or the Fabric8 Launcher tool.

4.6.7.2. Authenticating HTTP requests using the command line

Request a token by sending a HTTP POST request to the Red Hat SSO server. In the following example, the jq CLI tool is used to extract the token value from the JSON response.

Prerequisites

- The secured example endpoint URL. For more information, see Section 4.6.7.1, "Getting the Secured example application API endpoint".
- The **jq** command-line tool (optional). To download the tool and for more information, see https://stedolan.github.io/jq/.

Procedure

1. Request an access token with **curl**, the credentials, and **<SSO_AUTH_SERVER_URL>** and extract the token from the response with the **jq** command:

```
curl -sk -X POST https://<SSO_AUTH_SERVER_URL>/auth/realms/master/protocol/openid-
connect/token \
  -d grant_type=password \
  -d username=alice\
  -d password=password \
  -d client_id=demoapp \
  -d client_secret=1daa57a2-b60e-468b-a3ac-25bd2dc2eadc \
  | jq -r '.access_token'

eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJRek1nbXhZMUhrQnpxTnR0SnkwMm5j
NTNtMGNiWDQxV1hNSTU1MFo4MGVBIn0.eyJqdGkiOiI0NDA3YTliNC04YWRhLTRlMTctOD
Q2ZS03YjI5MjMyN2RmYTIiLCJleHAiOjE1MDc3OTM3ODcsIm5iZiI6MCwiaWF0IjoxNTA3Nzkz
NzI3LCJpc3MiOiJodHRwczovL3NlY3VyZS1zc28tc3NvLWRlbW8uYXBwcy5jYWZlLWJhYmUub
3JnL2F1dGgvcmVhbG1zL21hc3RlciIsImF1ZCI6ImRlbW9hcHAiLCJzdWIiOiJjMDE3NWNjYi0w
ODkyLTRiMzEtODI5Zi1kZGE4NzM4MTVmZTgiLCJ0eXAiOiJCZWFyZXIiLCJhenAiOiJkZW1vY
XBwIiwiYXV0aF90aW1lIjowLCJzZXNzaW9uX3N0YXRlIjoiMDFjOTkzNGQtNmZmOS00NWYzL
WJkNWUtMTU4NDI5ZDZjNDczIiwiYWNyIjoiMSIsImNsaWVudF9zZXNzaW9uIjoiMzM3Yzk0MT
YtYTdlZS00ZWUzLThjZWQtODhlODI0MGJjNTAyIiwiYWxsb3dlZC1vcmlnaW5zIjpbIioiXSwicmV
hbG1fYWNjZXNzIjp7InJvbGVzIjpbImJvb3N0ZXItYWRtaW4iXX0sInJlc291cmNlX2FjY2VzcyI6ey
JzZWN1cmVkLWJvb3N0ZXItZW5kcG9pbnQiOnsicm9sZXMiOlsiYm9vc3Rlci1hZG1pbiJdfSwiY
WNjb3VudCI6eyJyb2xlcyI6WyJtYW5hZ2UtYWNjb3VudCIsInZpZXctcHJvZmlsZSJdfX0sIm5hbW
UiOiJBbGljZSBJbkNoYWlucyIsInByZWZlcnJlZF91c2VybmFtZSI6ImFsaWNlIiwiZ2l2ZW5fbmFtZ
SI6IkFsaWNlIiwiZmFtaWx5X25hbWUiOiJJbkNoYWlucyIsImVtYWlsIjoiYWxpY2VAa2V5Y2xvYW
sub3JnIn0.mjmZe37enHpigJv0BGuIitOj-
kfMLPNwYzNd3n0Ax4Nga7KpnfytGyuPSvR4KAG8rzkfBNN9klPYdy7pJEeYlfmnFUkM4EDrZY
gn4qZAznP1Wzy1RfVRdUFi0-
GqFTMPb37o5HRldZZ09QljX_j3GHnoMGXRtYW9RZN4eKkYkcz9hRwgfJoTy2CuwFqeJwZY
UyXifrfA-JoTr0UmSUed-0NMksGrtJjjPggUGS-
qOn6OgKcmN2vaVAQlxW32y53JqUXctfLQ6DhJzIMYTmOflIPy0sgG1mG7sovQhw1xTg0vTjdx
8zQ-EJcexkj7IivRevRZsslKgqRFWs67jQAFQA
```

<SSO_AUTH_SERVER_URL> is the url of the **secure-sso** service.

The attributes, such as **username**, **password**, and **client_secret** are usually kept secret, but the above command uses the default provided credentials with this example for demonstration purpose.

If you do not want to use **jq** to extract the token, you can run just the **curl** command and manually extract the access token.

NOTE

The **-sk** option tells curl to ignore failures resulting from self-signed certificates. Do not use this option in a production environment. On macOS, you must have **curl** version **7.56.1** or greater installed. It must also be built with OpenSSL.

1. Invoke the Secured service. Attach the access (bearer) token to the HTTP headers:

```
$ curl -v -H "Authorization: Bearer <TOKEN>" http://<SERVICE_HOST>/api/greeting

{
   "content": "Hello, World!",
   "id": 2
}
```

Example 4.2. A sample GET Request Headers with an Access (Bearer) Token

```
> GET /api/greeting HTTP/1.1
> Host: <SERVICE_HOST>
> User-Agent: curl/7.51.0
> Accept: */*
> Authorization: Bearer <TOKEN>
```

<SERVICE_HOST> is the URL of the secured example endpoint. For more information, see Section 4.6.7.1, "Getting the Secured example application API endpoint" .

2. Verify the signature of the access token.
The access token is a JSON Web Token, so you can decode it using the JWT Debugger:

    a. In a web browser, navigate to the JWT Debugger website.

    b. Select **RS256** from the *Algorithm* drop down menu.

    NOTE

    Make sure the web form has been updated after you made the selection, so it displays the correct RSASHA256(...) information in the Signature section. If it has not, try switching to HS256 and then back to RS256.

    c. Paste the following content in the topmost text box into the *VERIFY SIGNATURE* section:

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAoETnPmN55xBJjRzN/cs30OzJ
9olkteLVNRjzdTxFOyRtS2ovDfzdhhO9XzUcTMbIsCOAZtSt8K+6yvBXypOSYvI75EUdypm
kcK1KoptqY5KEBQ1KwhWuP7IWQ0fshUwD6jI1QWDfGxfM/h34FvEn/0tJ71xN2P8TI2Yan
wuDZgosdobx/PAvlGREBGuk4BgmexTOkAdnFxIUQcCkiEZ2C41uCrxiS4CEe5OX91aK9
HKZV4ZJX6vnqMHmdDnsMdO+UFtxOBYZio+a1jP4W3d7J5fGeiOaXjQCOpivKnP2yU2D
PdWmDMyVb67l8DRA+jh0OJFKZ5H2fNgE3Il59vdsRwIDAQAB
-----END PUBLIC KEY-----
```

NOTE

This is the master realm public key from the Red Hat SSO server deployment of the Secured example application.

d. Paste the **token** output from the client output into the *Encoded* box.
The *Signature Verified* sign is displayed on the debugger page.

4.6.7.3. Authenticating HTTP requests using the web interface

In addition to the HTTP API, the secured endpoint also contains a web interface to interact with.

The following procedure is an exercise for you to see how security is enforced, how you authenticate, and how you work with the authentication token.

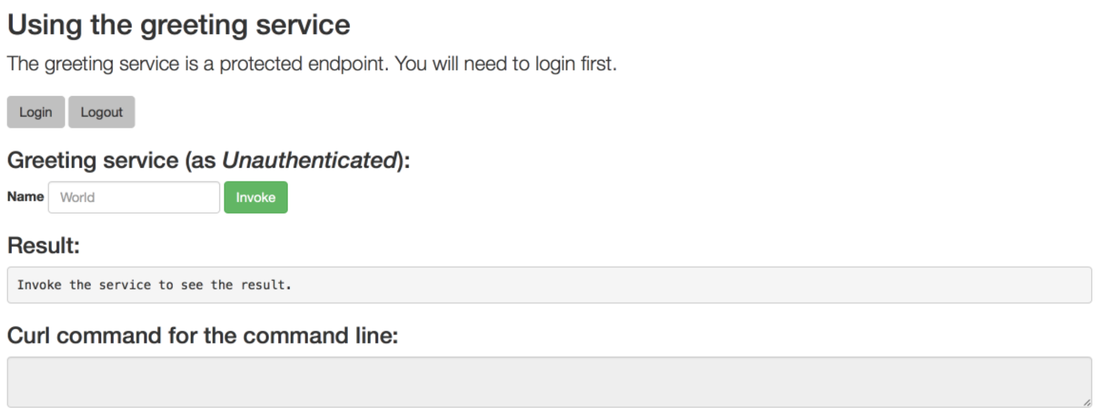
Prerequisites

- The secured endpoint URL. For more information, see Section 4.6.7.1, “Getting the Secured example application API endpoint”.

Procedure

1. In a web browser, navigate to the endpoint URL.
2. Perform an unauthenticated request:
   a. Click the *Invoke* button.

Figure 4.1. Unauthenticated Secured Example Web Interface



Using the greeting service

The greeting service is a protected endpoint. You will need to login first.

Greeting service (as *Unauthenticated*):

Result:

```
Invoke the service to see the result.
```

Curl command for the command line:

The services responds with an **HTTP 403 Forbidden** status code.

NOTE

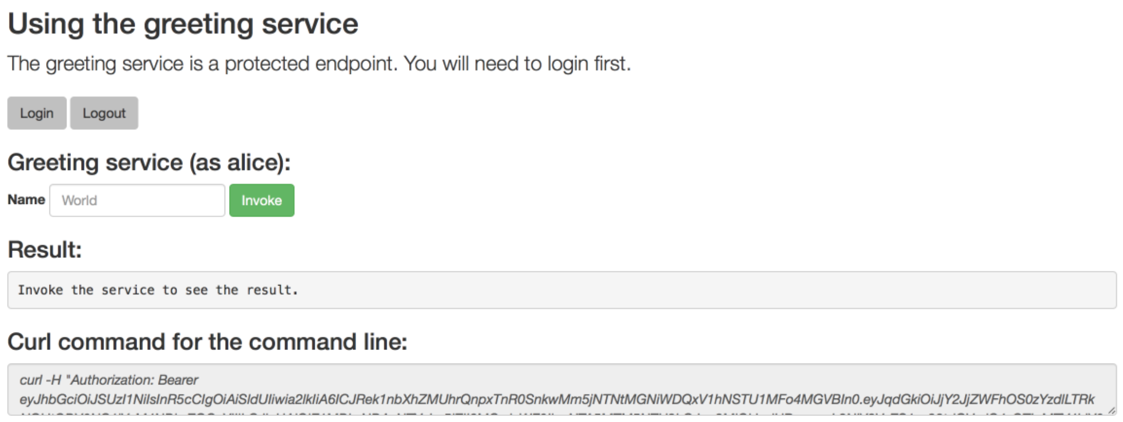
This is not the correct status code. It should be **HTTP 401 Unauthorized**. This issue has been identified and this example will be updated as soon as it is resolved.

3. Perform an authenticated request as a user:
   a. Click the *Login* button to authenticate against Red Hat SSO. You will be redirected to the SSO server.

b. Log in as the *Alice* user. You will be redirected back to the web interface.

> **NOTE**
>
> You can see the access (bearer) token in the command line output at the bottom of the page.

Figure 4.2. Authenticated Secured Example Web Interface (as Alice)



Using the greeting service

The greeting service is a protected endpoint. You will need to login first.

Greeting service (as alice):

Result:

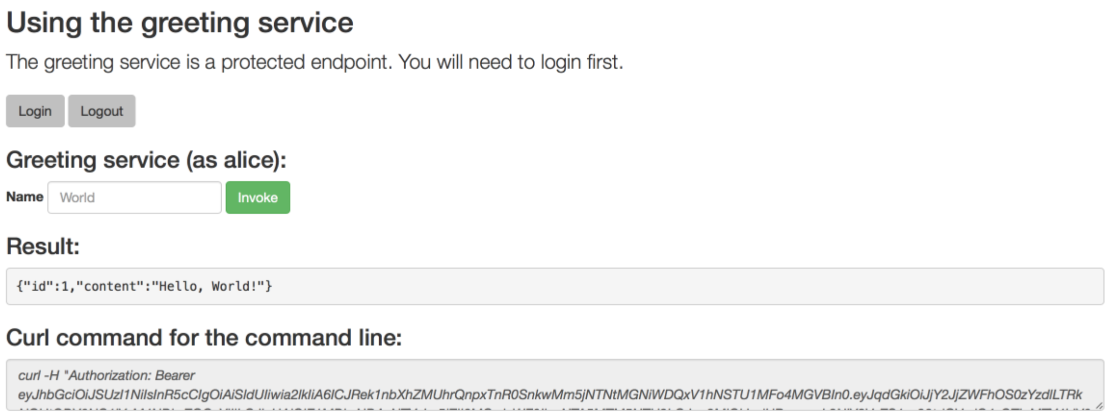
```
Invoke the service to see the result.
```

Curl command for the command line:

```
curl -H "Authorization: Bearer
eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJRek1nbXhZMUhrQnpxTnR0SnkwMm5jNTNtMGNiWDQxV1hNSTU1MFo4MGVBIn0.eyJqdGkiOiJjY2JjZWFhOS0zYzdlLTRk
```

c. Click *Invoke* again to access the Greeting service.
Confirm that there is no exception and the JSON response payload is displayed. This means the service accepted your access (bearer) token and you are authorized access to the Greeting service.

Figure 4.3. The Result of an Authenticated Greeting Request (as Alice)



Using the greeting service

The greeting service is a protected endpoint. You will need to login first.

Greeting service (as alice):

Result:

```
{"id":1,"content":"Hello, World!"}
```

Curl command for the command line:

```
curl -H "Authorization: Bearer
eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJRek1nbXhZMUhrQnpxTnR0SnkwMm5jNTNtMGNiWDQxV1hNSTU1MFo4MGVBIn0.eyJqdGkiOiJjY2JjZWFhOS0zYzdlLTRk
```

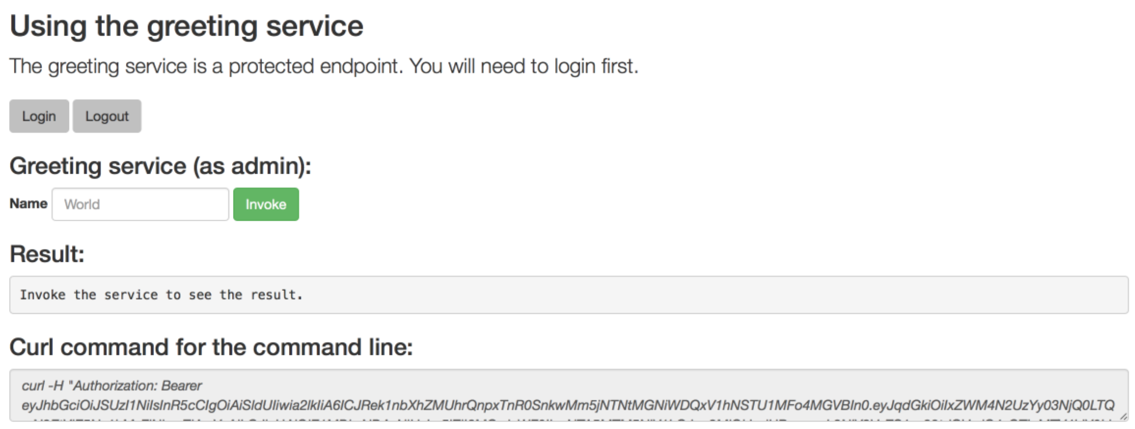
d. Log out.

4. Perform an authenticated request as an admininstrator:

a. Click the *Invoke* button.
Confirm that this sends an unauthenticated request to the Greeting service.

b. Click the *Login* button and log in as the *admin* user.

Figure 4.4. Authenticated Secured Example Web Interface (as admin)



Using the greeting service

The greeting service is a protected endpoint. You will need to login first.

Greeting service (as admin):

Name: World

Result:

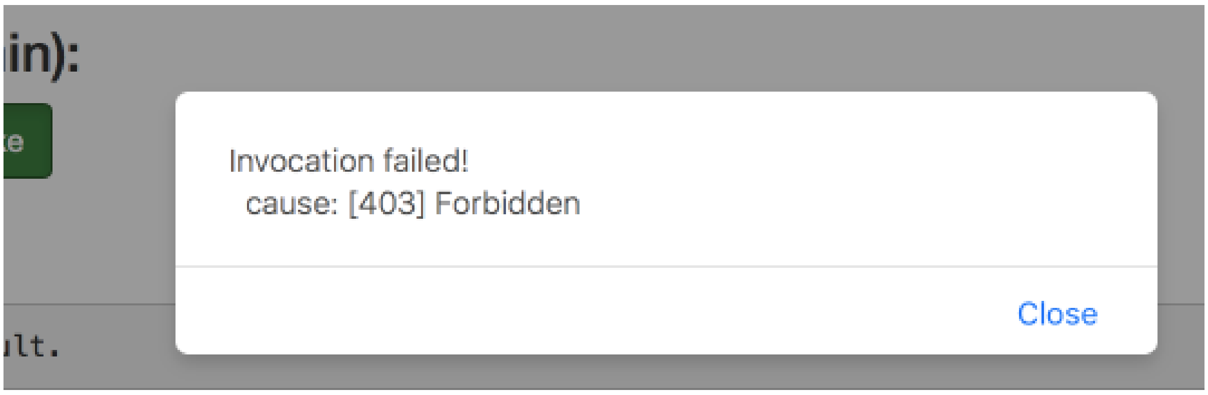
```
Invoke the service to see the result.
```

Curl command for the command line:

```
curl -H "Authorization: Bearer
eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJRek1nbXhZMUhrQnpxTnR0SnkwMm5jNTNtMGNiWDQxV1hNSTU1MFo4MGVBIn0.eyJqdGkiOiIxZWM4N2UzYy03NjQ0LTQ
```

5. Click the *Invoke* button.
   The service responds with an **HTTP 403 Forbidden** status code because the *admin* user is not authorized to access the Greeting service.

   Figure 4.5. Unauthorized Error Message

   Invocation failed!
   cause: [403] Forbidden

4.6.8. Secured SSO resources

Follow the links below for additional information on the principles behind the OAuth2 specification and on securing your applications using Red Hat SSO and Keycloak:

- Aaron Parecki: OAuth2 Simplified
- Red Hat SSO 7.1 Documentation
- Keycloak 3.2 Documentation
- Secured for Spring Boot
- Secured for Eclipse Vert.x
- Secured for Thorntail

4.7. CACHE EXAMPLE FOR NODE.JS

IMPORTANT

The following example is not meant to be run in a production environment.

Limitation: Run this example application on a Minishift or CDK. You can also use a manual workflow to deploy this example to OpenShift Online Pro and OpenShift Container Platform. This example is not currently available on OpenShift Online Starter.

Example proficiency level: **Advanced**.

The Cache example demonstrates how to use a cache to increase the response time of applications.

This example shows you how to:

- Deploy a cache to OpenShift.
- Use a cache within an application.

4.7.1. How caching works and when you need it

Caches allows you to store information and access it for a given period of time. You can access information in a cache faster or more reliably than repeatedly calling the original service. A disadvantage of using a cache is that the cached information is not up to date. However, that problem can be reduced by setting an *expiration* or TTL (time to live) on each value stored in the cache.

Example 4.3. Caching example

Assume you have two applications: *service1* and *service2*:

- *Service1* depends on a value from *service2*.
    - If the value from *service2* infrequently changes, *service1* could cache the value from *service2* for a period of time.
    - Using cached values can also reduce the number of times *service2* is called.
- If it takes *service1* 500 ms to retrieve the value directly from *service2*, but 100 ms to retrieve the cached value, *service1* would save 400 ms by using the cached value for each cached call.
- If *service1* would make uncached calls to *service2* 5 times per second, over 10 seconds, that would be 50 calls.
- If *service1* started using a cached value with a TTL of 1 second instead, that would be reduced to 10 calls over 10 seconds.

How the Cache example works

1. The *cache*, *cute name*, and *greeting* services are deployed and exposed.
2. User accesses the web frontend of the *greeting* service.
3. User invokes the *greeting* HTTP API using a button on the web frontend.
4. The *greeting* service depends on a value from the *cute name* service.
    - The *greeting* service first checks if that value is stored in the *cache* service. If it is, then the cached value is returned.
    - If the value is not cached, the *greeting* service calls the *cute name* service, returns the value, and stores the value in the *cache* service with a TTL of 5 seconds.

5. The web front end displays the response from the *greeting* service as well as the total time of the operation.

6. User invokes the service multiple times to see the difference between cached and uncached operations.

    - Cached operations are significantly faster than uncached operations.

    - User can force the cache to be cleared before the TTL expires.

4.7.2. Deploying the Cache example application to OpenShift Online

Use one of the following options to execute the Cache example application on OpenShift Online.

- Use developers.redhat.com/launch

- Use the **oc** CLI client

Although each method uses the same **oc** commands to deploy your application, using developers.redhat.com/launch provides an automated deployment workflow that executes the **oc** commands for you.

4.7.2.1. Deploying the example application using developers.redhat.com/launch

Prerequisites

- An account at OpenShift Online.

Procedure

1. Navigate to the developers.redhat.com/launch URL in a browser and log in.

2. Follow on-screen instructions to create and launch your example application in Node.js.

4.7.2.2. Authenticating the oc CLI client

To work with example applications on OpenShift Online using the **oc** command-line client, you must authenticate the client using the token provided by the OpenShift Online web interface.

Prerequisites

- An account at OpenShift Online.

Procedure

1. Navigate to the OpenShift Online URL in a browser.

2. Click on the question mark icon in the top right-hand corner of the Web console, next to your user name.

3. Select *Command Line Tools* in the drop-down menu.

4. Find the text box that contains the **oc login …** command with the hidden token, and click the button next to it to copy its content to your clipboard.

5. Paste the command into a terminal application. The command uses your authentication token to authenticate your **oc** CLI client with your OpenShift Online account.

```
$ oc login OPENSHIFT_URL --token=MYTOKEN
```

4.7.2.3. Deploying the Cache example application using the oc CLI client

Prerequisites

- The example application created using developers.redhat.com/launch. For more information, see Section 4.7.2.1, "Deploying the example application using developers.redhat.com/launch" .
- The **oc** client authenticated. For more information, see Section 4.7.2.2, "Authenticating the **oc** CLI client".

Procedure

1. Clone your project from GitHub.

```
$ git clone git@github.com:USERNAME/MY_PROJECT_NAME.git
```

Alternatively, if you downloaded a ZIP file of your project, extract it.

```
$ unzip MY_PROJECT_NAME.zip
```

2. Create a new project.

```
$ oc new-project MY_PROJECT_NAME
```

3. Navigate to the root directory of your application.

4. Deploy the cache service.

```
$ oc apply -f service.cache.yml
```

NOTE

If you are using an architecture other than x86_64, in the YAML file, update the image name of Red Hat Data Grid to its relevant image name in that architecture. For example, for the s390x architecture, update the image name to its IBM Z image name **registry.access.redhat.com/jboss-datagrid-7/datagrid73-openj9-11-openshift-rhel8**.

5. Use **start-openshift.sh** to start the deployment to OpenShift.

```
$ ./start-openshift.sh
```

6. Check the status of your application and ensure your pod is running.

```
$ oc get pods -w
NAME                         READY     STATUS      RESTARTS   AGE
cache-server-123456789-aaaaa           1/1       Running     0          8m
```

```
MY_APP_NAME-cutename-1-bbbbb       1/1       Running     0         4m
MY_APP_NAME-cutename-s2i-1-build   0/1       Completed   0         7m
MY_APP_NAME-greeting-1-ccccc       1/1       Running     0         3m
MY_APP_NAME-greeting-s2i-1-build   0/1       Completed   0         3m
```

Your 3 pods should have a status of **Running** once they are fully deployed and started.

7. After your example application is deployed and started, determine its route.

Example Route Information

```
$ oc get routes
NAME              HOST/PORT                                          PATH      SERVICES
PORT      TERMINATION
MY_APP_NAME-cutename   MY_APP_NAME-cutename-
MY_PROJECT_NAME.OPENSHIFT_HOSTNAME           MY_APP_NAME-cutename   8080
None
MY_APP_NAME-greeting   MY_APP_NAME-greeting-
MY_PROJECT_NAME.OPENSHIFT_HOSTNAME           MY_APP_NAME-greeting   8080
None
```

The route information of a pod gives you the base URL which you use to access it. In the example above, you would use **http://MY_APP_NAME-greeting-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME** as the base URL to access the greeting service.

4.7.3. Deploying the Cache example application to Minishift or CDK

Use one of the following options to execute the Cache example application locally on Minishift or CDK:

- Using Fabric8 Launcher
- Using the **oc** CLI client

Although each method uses the same **oc** commands to deploy your application, using Fabric8 Launcher provides an automated deployment workflow that executes the **oc** commands for you.

4.7.3.1. Getting the Fabric8 Launcher tool URL and credentials

You need the Fabric8 Launcher tool URL and user credentials to create and deploy example applications on Minishift or CDK. This information is provided when the Minishift or CDK is started.

Prerequisites

- The Fabric8 Launcher tool installed, configured, and running.

Procedure

1. Navigate to the console where you started Minishift or CDK.
2. Check the console output for the URL and user credentials you can use to access the running Fabric8 Launcher:

Example Console Output from a Minishift or CDK Startup

```
...
-- Removing temporary directory ... OK
-- Server Information ...
   OpenShift server started.
   The server is accessible via web console at:
       https://192.168.42.152:8443

   You are logged in as:
       User:     developer
       Password: developer

   To login as administrator:
       oc login -u system:admin
```

4.7.3.2. Deploying the example application using the Fabric8 Launcher tool

Prerequisites

- The URL of your running Fabric8 Launcher instance and the user credentials of your Minishift or CDK. For more information, see Section 4.7.3.1, "Getting the Fabric8 Launcher tool URL and credentials".

Procedure

1. Navigate to the Fabric8 Launcher URL in a browser.
2. Follow the on-screen instructions to create and launch your example application in Node.js.

4.7.3.3. Authenticating the oc CLI client

To work with example applications on Minishift or CDK using the **oc** command-line client, you must authenticate the client using the token provided by the Minishift or CDK web interface.

Prerequisites

- The URL of your running Fabric8 Launcher instance and the user credentials of your Minishift or CDK. For more information, see Section 4.7.3.1, "Getting the Fabric8 Launcher tool URL and credentials".

Procedure

1. Navigate to the Minishift or CDK URL in a browser.
2. Click on the question mark icon in the top right-hand corner of the Web console, next to your user name.
3. Select *Command Line Tools* in the drop-down menu.
4. Find the text box that contains the **oc login …** command with the hidden token, and click the button next to it to copy its content to your clipboard.
5. Paste the command into a terminal application. The command uses your authentication token to authenticate your **oc** CLI client with your Minishift or CDK account.

```
$ oc login OPENSHIFT_URL --token=MYTOKEN
```

4.7.3.4. Deploying the Cache example application using the oc CLI client

Prerequisites

- The example application created using Fabric8 Launcher tool on a Minishift or CDK. For more information, see Section 4.7.3.2, “Deploying the example application using the Fabric8 Launcher tool”.
- Your Fabric8 Launcher tool URL.
- The **oc** client authenticated. For more information, see Section 4.7.3.3, “Authenticating the **oc** CLI client”.

Procedure

1. Clone your project from GitHub.

   ```
   $ git clone git@github.com:USERNAME/MY_PROJECT_NAME.git
   ```

   Alternatively, if you downloaded a ZIP file of your project, extract it.

   ```
   $ unzip MY_PROJECT_NAME.zip
   ```

2. Create a new project.

   ```
   $ oc new-project MY_PROJECT_NAME
   ```

3. Navigate to the root directory of your application.
4. Deploy the cache service.

   ```
   $ oc apply -f service.cache.yml
   ```

   NOTE

   If you are using an architecture other than x86_64, in the YAML file, update the image name of Red Hat Data Grid to its relevant image name in that architecture. For example, for the s390x architecture, update the image name to its IBM Z image name **registry.access.redhat.com/jboss-datagrid-7/datagrid73-openj9-11-openshift-rhel8**.

5. Use **start-openshift.sh** to start the deployment to OpenShift.

   ```
   $ ./start-openshift.sh
   ```

6. Check the status of your application and ensure your pod is running.

   ```
   $ oc get pods -w
   NAME                          READY     STATUS      RESTARTS   AGE
   cache-server-123456789-aaaaa          1/1       Running     0          8m
   ```

```
MY_APP_NAME-cutename-1-bbbbb       1/1       Running     0         4m
MY_APP_NAME-cutename-s2i-1-build   0/1       Completed   0         7m
MY_APP_NAME-greeting-1-ccccc       1/1       Running     0         3m
MY_APP_NAME-greeting-s2i-1-build   0/1       Completed   0         3m
```

Your 3 pods should have a status of **Running** once they are fully deployed and started.

7. After your example application is deployed and started, determine its route.

   Example Route Information

```
$ oc get routes
NAME              HOST/PORT                                           PATH      SERVICES
PORT      TERMINATION
MY_APP_NAME-cutename   MY_APP_NAME-cutename-
MY_PROJECT_NAME.OPENSHIFT_HOSTNAME            MY_APP_NAME-cutename   8080
None
MY_APP_NAME-greeting   MY_APP_NAME-greeting-
MY_PROJECT_NAME.OPENSHIFT_HOSTNAME            MY_APP_NAME-greeting   8080
None
```

   The route information of a pod gives you the base URL which you use to access it. In the example above, you would use **http://MY_APP_NAME-greeting-MY_PROJECT_NAME.OPENSHIFT_HOSTNAME** as the base URL to access the greeting service.

4.7.4. Deploying the Cache example application to OpenShift Container Platform

The process of creating and deploying example applications to OpenShift Container Platform is similar to OpenShift Online:

Prerequisites

- The example application created using developers.redhat.com/launch.

Procedure

- Follow the instructions in Section 4.7.2, "Deploying the Cache example application to OpenShift Online", only use the URL and user credentials from the OpenShift Container Platform Web Console.

4.7.5. Interacting with the unmodified Cache example application

Prerequisites

- Your application deployed

Procedure

1. Navigate to the **greeting** service using your browser.
2. Click *Invoke the service* once.
   Notice the **duration** value is above **2000**. Also notice the cache state has changed form **No cached value** to **A value is cached**.

3. Wait 5 seconds and notice cache state has changed back to **No cached value**.
   The TTL for the cached value is set to 5 seconds. When the TTL expires, the value is no longer cached.

4. Click *Invoke the service* once more to cache the value.

5. Click *Invoke the service* a few more times over the course of a few seconds while cache state is **A value is cached**.
   Notice a significantly lower **duration** value since it is using a cached value. If you click *Clear the cache*, the cache is emptied.

4.7.6. Caching resources

More background and related information on caching can be found here:

- Cache for Spring Boot
- Cache for Eclipse Vert.x
- Cache for Thorntail

CHAPTER 5. DEBUGGING YOUR NODE.JS BASED APPLICATION

This section contains information about debugging your Node.js–based application and using debug logging in both local and remote deployments.

5.1. REMOTE DEBUGGING

To remotely debug an application, you need to start it in a debugging mode and attach a debugger to it.

5.1.1. Starting your application locally and attaching the native debugger

The native debugger enables you to debug your Node.js–based application using the built-in debugging client.

Prerequisites

- An application you want to debug.

Procedure

1. Start the application with the debugger enabled.
   The native debugger is automatically attached and provides a debugging prompt.

   Example application with the debugger enabled

   ```
   $ node inspect app.js
   < Debugger listening on ws://127.0.0.1:9229/12345678-aaaa-bbbb-cccc-0123456789ab
   < For help see https://nodejs.org/en/docs/inspector
   < Debugger attached.
   ...
   debug>
   ```

   If you have a different entry point for your application, you need to change the command to specify that entry point:

   ```
   $ node inspect path/to/entrypoint
   ```

   For example, when using the express generator to create your application, the entry point is set to **./bin/www** by default. Some of the examples, such as the REST API Level 0 example application, use **./bin/www** as an entry point.

2. Use the debugger prompt to perform debugging commands.

5.1.2. Starting your application locally and attaching the V8 inspector

The V8 inspector enables you to debug your Node.js–based application using other tools, such as Chrome DevTools, that use the Chrome Debugging Protocol.

Prerequisites

- An application you want to debug.

- The V8 inspector installed, such as the one provided in the Google Chrome Browser .

Procedure

1. Start your application with the V8 inspector integration enabled.

   ```
   $ node --inspect app.js
   ```

   If you have a different entry point for your application, you need to change the command to specify that entry point:

   ```
   $ node --inspect path/to/entrypoint
   ```

   For example, when using the express generator to create your application, the entry point is set to **./bin/www** by default. Some of the examples, such as the REST API Level 0 example application, use **./bin/www** as an entry point.

2. Attach the V8 inspector and perform debugging commands.
   For example, if using Google Chrome:

   a. Navigate to **chrome://inspect**.

   b. Select your application from below *Remote Target*.

   c. You can now see the source of your application and can perform debugging actions.

5.1.3. Starting your application on OpenShift in debugging mode

To debug your Node.js-based application on OpenShift remotely, you must set the **NODE_ENV** environment variable inside the container to **development** and configure port forwarding so that you can connect to your application from a remote debugger.

Prerequisites

- Your application running on OpenShift.
- The **oc** binary installed on your machine.
- The ability to execute the **oc port-forward** command in your target OpenShift environment.

Procedure

1. Using the **oc** command, list the available deployment configurations:

   ```
   $ oc get dc
   ```

2. Set the **NODE_ENV** environment variable in the deployment configuration of your application to **development** to enable debugging. For example:

   ```
   $ oc set env dc/MY_APP_NAME NODE_ENV=development
   ```

3. Redeploy the application if it is not set to redeploy automatically on configuration change. For example:

```
$ oc rollout latest dc/MY_APP_NAME
```

4. Configure port forwarding from your local machine to the application pod:

    a. List the currently running pods and find one containing your application:

    ```
    $ oc get pod
    NAME                    READY    STATUS     RESTARTS   AGE
    MY_APP_NAME-3-1xrsp       0/1      Running     0         6s
    ...
    ```

    b. Configure port forwarding:

    ```
    $ oc port-forward MY_APP_NAME-3-1xrsp $LOCAL_PORT_NUMBER:5858
    ```

    Here, **$LOCAL_PORT_NUMBER** is an unused port number of your choice on your local machine. Remember this number for the remote debugger configuration.

5. Attach the V8 inspector and perform debugging commands.
For example, if using Google Chrome:

    a. Navigate to **chrome://inspect**.

    b. Click *Configure*.

    c. Add **127.0.0.1:$LOCAL_PORT_NUMBER**.

    d. Click *Done*.

    e. Select your application from below *Remote Target*.

    f. You can now see the source of your application and can perform debugging actions.

6. When you are done debugging, unset the **NODE_ENV** environment variable in your application pod. For example:

```
$ oc set env dc/MY_APP_NAME NODE_ENV-
```

5.2. DEBUG LOGGING

Debug logging is a way to add detailed information to the application log when debugging. This allows you to:

- Keep minimal logging output during normal operation of the application for improved readability and reduced disk space usage.

- View detailed information about the inner workings of the application when resolving issues.

5.2.1. Add debug logging

This example uses the debug package, but there are also other packages available that can handle debug logging.

Prerequisites

- An application you want to debug. For example, an example.

Procedure

1. Add the **debug** logging definition.

```
const debug = require('debug')('myexample');
```

2. Add debug statements.

```
app.use('/api/greeting', (request, response) => {
  const name = request.query ? request.query.name : undefined;
  //log name in debugging
  debug('name: '+name);
  response.send({content: `Hello, ${name || 'World'}`});
});
```

3. Add the debug module to **package.json**.

```
...
"dependencies": {
    "debug": "^3.1.0"
  }
```

   Depending on your application, this module may already be included. For example, when using the express generator to create your application, the **debug** module is already added to **package.json**. Some of the example applications, such as the REST API Level 0 example, already have the **debug** module in the **package.json** file.

4. Install the application dependencies.

```
$ npm install
```

5.2.2. Accessing debug logs on localhost

Use the **DEBUG** environment variable when starting your application to enable debug logging.

Prerequisites

- An application with debug logging.

Procedure

1. Set the **DEBUG** environment variable when starting your application to enable debug logging.

```
$ DEBUG=myexample npm start
```

   The **debug** module can use wildcards to filter debugging messages. This is set using the **DEBUG** environment variable.

2. Test your application to invoke debug logging.
   For example, when debug logging in the REST API Level 0 example is set to log the **name** variable in the **/api/greeting** method:

```
$ curl http://localhost:8080/api/greeting?name=Sarah
```

3. View your application logs to see your debug messages.

```
myexample name: Sarah +3m
```

5.2.3. Accessing Node.js debug logs on OpenShift

Use the the **DEBUG** environment variable in your application pod in OpenShift to enable debug logging.

Prerequisites

- An application with debug logging.
- The **oc** CLI client installed.

Procedure

1. Use the **oc** CLI client to log into your OpenShift instance.

```
$ oc login ...
```

2. Deploy your application to OpenShift.

```
$ npm run openshift
```

This runs the **openshift** npm script, which wraps direct calls to nodeshift.

3. Find the name of your pod and follow the logs to watch it start.

```
$ oc get pods
....
$ oc logs -f pod/POD_NAME
```

IMPORTANT

After your pod has started, leave this command running and execute the remaining steps in a new terminal window. This allows you to *follow* the logs and see new entries made to it.

4. Test your application.
For example, if you had debug logging in the REST API Level 0 example to log the **name** variable in the **/api/greeting** method:

```
$ oc get routes
...
$ curl $APPLICATION_ROUTE/api/greeting?name=Sarah
```

5. Return to your pod logs and notice there are no debug logging messages in the logs.

6. Set the **DEBUG** environment variable to enable debug logging.

```
$ oc get dc
...
$ oc set env dc DC_NAME DEBUG=myexample
```

7. Return to your pod logs to watch the update roll out.
   After the update has rolled out, your pod will stop and you will no longer be following the logs.

8. Find the name of your new pod and follow the logs.

   ```
   $ oc get pods
   ....
   $ oc logs -f pod/POD_NAME
   ```

   IMPORTANT

   After your pod has started, leave this command running and execute the remaining steps in a different terminal window. This allows you to *follow* the logs and see new entries made to it. Specifically, the logs will show your debug messages.

9. Test the application to invoke debug logging.

   ```
   $ oc get routes
   ...
   $ curl $APPLICATION_ROUTE/api/greeting?name=Sarah
   ```

10. Return to your pod logs to see the debug messages.

    ```
    ...
    myexample name: Sarah +3m
    ```

To disable debug logging, remove the **DEBUG** environment variable from the pod:

```
$ oc set env dc DC_NAME DEBUG-
```

Additional resources

More details on environment variables are available in the OpenShift documentation.

CHAPTER 6. DEVELOPING AND DEPLOYING A NODE.JS APPLICATION

In addition to using an example , you can create new Node.js applications from scratch and deploy them to OpenShift.

6.1. DEVELOPING A NODE.JS APPLICATION

For a basic Node.js application, you must create a JavaScript file containing Node.js methods.

Prerequisites

- **npm** installed.

Procedure

1. Create a new directory **myApp**, and navigate to it.

   ```
   $ mkdir myApp
   $ cd MyApp
   ```

   This is the root directory for the application.

2. Initialize your application with **npm**.
   The rest of this example assumes the entry point is **app.js**, which you are prompted to set when running **npm init**.

   ```
   $ cd myApp
   $ npm init
   ```

3. Create the entry point in a new file called **app.js**.

   Example app.js

   ```
   const http = require('http');

   const server = http.createServer((request, response) => {
     response.statusCode = 200;
     response.setHeader('Content-Type', 'application/json');

     const greeting = {content: 'Hello, World!'};

     response.write(JSON.stringify(greeting));
     response.end();
   });

   server.listen(8080, () => {
     console.log('Server running at http://localhost:8080');
   });
   ```

4. Start your application.

```
$ node app.js
Server running at http://localhost:8080
```

5. Using **curl** or your browser, verify your application is running at **http://localhost:8080**.

```
$ curl http://localhost:8080
{"content":"Hello, World!"}
```

Additional information

- The Node.js runtime provides the core Node.js API which is documented in the Node.js API documentation.

6.2. DEPLOYING A NODE.JS APPLICATION TO OPENSHIFT

To deploy your Node.js application to OpenShift, add **nodeshift** to the application, configure the **package.json** file and then deploy using **nodeshift**.

6.2.1. Preparing Node.js application for OpenShift deployment

To prepare a Node.js application for OpenShift deployment, you must perform the following steps:

- Add **nodeshift** to the application.
- Add **openshift** and **start** entries to the **package.json** file.

Prerequisites

- **npm** installed.

Procedure

1. Add **nodeshift** to your application.

```
$ npm install nodeshift --save-dev
```

2. Add the **openshift** and **start** entries to the **scripts** section in **package.json**.

```
{
  "name": "myApp",
  "version": "1.0.0",
  "description": "",
  "main": "app.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "openshift": "nodeshift --expose --
dockerImage=registry.access.redhat.com/rhscl/ubi8/nodejs-10",
    "start": "node app.js",
    ...
  }
  ...
}
```

The **openshift** script uses **nodeshift** to deploy the application to OpenShift.

> **NOTE**
>
> Universal base images and RHEL images are available for Node.js. See the Node.js release notes for more information on image names.

3. *Optional*: Add a **files** section in **package.json**.

```
{
  "name": "myApp",
  "version": "1.0.0",
  "description": "",
  "main": "app.js",
  "scripts": {
    ...
  },
  "files": [
    "package.json",
    "app.js"
  ]
  ...
}
```

The **files** section tells **nodeshift** what files and directories to include when deploying to OpenShift. **nodeshift** uses the **node-tar** module to create a tar file based on the files and directories you list in the **files** section. This tar file is used when **nodeshift** deploys your application to OpenShift. If the **files** section is not specified, **nodeshift** will send the entire current directory, excluding:

- **node_modules/**
- **.git/**
- **tmp/**
  It is recommended that you include a **files** section in **package.json** to avoid including unnecessary files when deploying to OpenShift.

6.2.2. Deploying a Node.js application to OpenShift

You can deploy a Node.js application to OpenShift using **nodeshift**.

Prerequisites

- The **oc** CLI client installed.
- **npm** installed.
- Ensure all the ports used by your application are correctly exposed when configuring your routes.

Procedure

1. Log in to your OpenShift instance with the **oc** client.

```
$ oc login ...
```

2. Use **nodeshift** to deploy the application to OpenShift.

```
$ npm run openshift
```

6.3. DEPLOYING A NODE.JS APPLICATION TO STAND-ALONE RED HAT ENTERPRISE LINUX

You can deploy a Node.js application to stand-alone Red Hat Enterprise Linux using **npm**.

Prerequisites

- A Node.js application.
- npm 6.4.1 installed
- RHEL 7 or RHEL 8 installed.
- Node.js installed

Procedure

1. If you have specified additional dependencies in the **package.json** file of your project, ensure that you install them before running your applications.

```
$ npm install
```

2. Deploy the application from the application’s root directory.

```
$ node app.js
Server running at http://localhost:8080
```

Verification steps

1. Use **curl** or your browser to verify your application is running at **http://localhost:8080**

```
$ curl http://localhost:8080
```

APPENDIX A. ABOUT NODESHIFT

Nodeshift is a module for running OpenShift deployments with Node.js projects.

IMPORTANT

Nodeshift assumes you have the **oc** CLI client installed, and you are logged into your OpenShift cluster. Nodeshift also uses the current project the **oc** CLI client is using.

Nodeshift uses resource files in the **.nodeshift** folder located at the root of the project to handle creating OpenShift Routes, Services and DeploymentConfigs. More details on Nodeshift are available on the Nodeshift project page .

APPENDIX B. UPDATING THE DEPLOYMENT CONFIGURATION OF AN EXAMPLE APPLICATION

The deployment configuration for an example application contains information related to deploying and running the application in OpenShift, such as route information or readiness probe location. The deployment configuration of an example application is stored in a set of YAML files. For examples that use the Fabric8 Maven Plugin, the YAML files are located in the **src/main/fabric8/** directory. For examples using Nodeshift, the YAML files are located in the **.nodeshift** directory.

IMPORTANT

The deployment configuration files used by the Fabric8 Maven Plugin and Nodeshift do not have to be full OpenShift resource definitions. Both Fabric8 Maven Plugin and Nodeshift can take the deployment configuration files and add some missing information to create a full OpenShift resource definition. The resource definitions generated by the Fabric8 Maven Plugin are available in the **target/classes/META-INF/fabric8/** directory. The resource definitions generated by Nodeshift are available in the **tmp/nodeshift/resource/** directory.

Prerequisites

- An existing example project.
- The **oc** CLI client installed.

Procedure

1. Edit an existing YAML file or create an additional YAML file with your configuration update.
   - For example, if your example already has a YAML file with a **readinessProbe** configured, you could change the **path** value to a different available path to check for readiness:

     ```
     spec:
       template:
         spec:
           containers:
             readinessProbe:
               httpGet:
                 path: /path/to/probe
                 port: 8080
                 scheme: HTTP
     ...
     ```

   - If a **readinessProbe** is not configured in an existing YAML file, you can also create a new YAML file in the same directory with the **readinessProbe** configuration.
2. Deploy the updated version of your example using Maven or npm.
3. Verify that your configuration updates show in the deployed version of your example.

   ```
   $ oc export all --as-template='my-template'

   apiVersion: v1
   kind: Template
   ```

```
metadata:
  creationTimestamp: null
  name: my-template
objects:
- apiVersion: v1
  kind: DeploymentConfig
  ...
  spec:
    ...
    template:
      ...
      spec:
        containers:
          ...
          livenessProbe:
            failureThreshold: 3
            httpGet:
              path: /path/to/different/probe
              port: 8080
              scheme: HTTP
            initialDelaySeconds: 60
            periodSeconds: 30
            successThreshold: 1
            timeoutSeconds: 1
          ...
```

Additional resources

If you updated the configuration of your application directly using the web-based console or the **oc** CLI client, export and add these changes to your YAML file. Use the **oc export all** command to show the configuration of your deployed application.

APPENDIX C. CONFIGURING A JENKINS FREESTYLE PROJECT TO DEPLOY YOUR NODE.JS APPLICATION WITH NODESHIFT

Similar to using nodeshift from your local host to deploy a Node.js application, you can configure Jenkins to use nodeshift to deploy a Node.js application.

Prerequisites

- Access to an OpenShift cluster.
- The Jenkins container image running on same OpenShift cluster.
- The Node.js plugin installed on your Jenkins server.
- A Node.js application configured to use nodeshift and the Red Hat base image.

  Example using the Red Hat base image with nodeshift

  ```
  $ nodeshift --dockerImage=registry.access.redhat.com/ubi8/nodejs-10 ...
  ```

- The source of the application available in GitHub.

Procedure

1. Create a new OpenShift project for your application:
   a. Open the OpenShift Web console and log in.
   b. Click *Create Project* to create a new OpenShift project.
   c. Enter the project information and click *Create*.
2. Ensure Jenkins has access to that project.
   For example, if you configured a service account for Jenkins, ensure that account has **edit** access to the project of your application.
3. Create a new freestyle Jenkins project on your Jenkins server:
   a. Click *New Item*.
   b. Enter a name, choose *Freestyle project*, and click *OK*.
   c. Under *Source Code Management*, choose *Git* and add the GitHub url of your application.
   d. Under *Build Environment*, make sure *Provide Node & npm bin/ folder to PATH* is checked and the Node.js environment is configured.
   e. Under *Build*, choose *Add build step* and select **Execute Shell**.
   f. Add the following to the *Command* area:

   ```
   npm install -g nodeshift
   nodeshift --dockerImage=registry.access.redhat.com/ubi8/nodejs-10 --
   namespace=MY_PROJECT
   ```

-

Substitute **MY_PROJECT** with the name of the OpenShift project for your application.

g. Click *Save*.

4. Click *Build Now* from the main page of the Jenkins project to verify your application builds and deploys to the OpenShift project for your application.
   You can also verify that your application is deployed by opening the route in the OpenShift project of the application.

Next steps

- Consider adding GITSCM polling or using the **Poll SCM** build trigger. These options enable builds to run every time a new commit is pushed to the GitHub repository.
- Consider adding nodeshift as a global package when configuring the Node.js plugin. This allows you to omit **npm install -g nodeshift** when adding your **Execute Shell** build step.
- Consider adding a build step that executes tests before deploying.

APPENDIX D. BREAKDOWN OF PACKAGE.JSON PROPERTIES

nodejs-rest-http/package.json

```
{
  "name": "nodejs-rest-http",
  "version": "1.1.1",
  "author": "Red Hat, Inc.",
  "license": "Apache-2.0",
  "scripts": {
    "test": "tape test/*.js | tap-spec", 1
    "lint": "eslint test/*.js app.js bin/*",
    "prepare": "nsp check",
    "coverage": "nyc npm test",
    "coveralls": "nyc npm test && nyc report --reporter=text-lcov | coveralls",
    "ci": "npm run lint && npm run coveralls",
    "dependencyCheck": "szero . --ci",
    "release": "standard-version",
    "openshift": "nodeshift --strictSSL=false --nodeVersion=8.x", 2
    "postinstall": "license-reporter report && license-reporter save --xml licenses.xml",
    "start": "node ." 3
  },
  "main": "./bin/www", 4
  "repository": {
    "type": "git",
    "url": "git://github.com/nodeshift-starters/nodejs-rest-http.git"
  },
  "files": [ 5
    "package.json",
    "app.js",
    "public",
    "bin",
    "LICENSE",
    "licenses"
  ],
  "bugs": {
    "url": "https://github.com/nodeshift-starters/nodejs-rest-http/issues"
  },
  "homepage": "https://github.com/nodeshift-starters/nodejs-rest-http",
  "devDependencies": { 6
    "coveralls": "^3.0.0",
    "nodeshift": "^1.3.0",
    "nsp": "~3.1.0",
    "nyc": "~11.4.1",
    "standard-version": "^4.2.0",
    "supertest": "^3.0.0",
    "szero": "^1.0.0",
    "tap-spec": "~4.1.1",
    "tape": "~4.8.0",
    "xo": "~0.20.3"
  },
  "dependencies": { 7
    "body-parser": "^1.18.2",
    "debug": "^3.1.0",
    "express": "^4.16.0",
    "license-reporter": "^1.1.3"
  }
}
```

1. A **npm** script for running unit tests. Run with **npm run test**.
2. A **npm** script for deploying this application to Minishift or CDK. Run with **npm run openshift**. The **strictSSL** option allows us to deploy to Minishift or CDK instances with self-signed certificates.
3. A **npm** script for starting this application. Run with **npm start**.
4. The primary entrypoint for the application when run with **npm start**.
5. Specifies the files to be included in the binary that is uploaded to Minishift or CDK.
6. A list of development dependencies to be installed from the **npm** registry. These are used for testing and deployment to Minishift or CDK.
7. A list of dependencies to be installed from the **npm** registry.

APPENDIX E. ADDITIONAL NODE.JS RESOURCES

- Node.js Home Page
- npm Home Page

APPENDIX F. APPLICATION DEVELOPMENT RESOURCES

For additional information about application development with OpenShift, see:

- OpenShift Interactive Learning Portal

To reduce network load and shorten the build time of your application, set up a Nexus mirror for Maven on your Minishift or CDK:

- Setting Up a Nexus Mirror for Maven

APPENDIX G. THE SOURCE-TO-IMAGE (S2I) BUILD PROCESS

Source-to-Image (S2I) is a build tool for generating reproducible Docker-formatted container images from online SCM repositories with application sources. With S2I builds, you can easily deliver the latest version of your application into production with shorter build times, decreased resource and network usage, improved security, and a number of other advantages. OpenShift supports multiple build strategies and input sources.

For more information, see the Source-to-Image (S2I) Build chapter of the OpenShift Container Platform documentation.

You must provide three elements to the S2I process to assemble the final container image:

- The application sources hosted in an online SCM repository, such as GitHub.
- The S2I Builder image, which serves as the foundation for the assembled image and provides the ecosystem in which your application is running.
- Optionally, you can also provide environment variables and parameters that are used by S2I scripts.

The process injects your application source and dependencies into the Builder image according to instructions specified in the S2I script, and generates a Docker-formatted container image that runs the assembled application. For more information, check the S2I build requirements, build options and how builds work sections of the OpenShift Container Platform documentation.

APPENDIX H. PROFICIENCY LEVELS

Each available example teaches concepts that require certain minimum knowledge. This requirement varies by example. The minimum requirements and concepts are organized in several levels of proficiency. In addition to the levels described here, you might need additional information specific to each example.

Foundational

The examples rated at Foundational proficiency generally require no prior knowledge of the subject matter; they provide general awareness and demonstration of key elements, concepts, and terminology. There are no special requirements except those directly mentioned in the description of the example.

Advanced

When using Advanced examples, the assumption is that you are familiar with the common concepts and terminology of the subject area of the example in addition to Kubernetes and OpenShift. You must also be able to perform basic tasks on your own, for example, configuring services and applications, or administering networks. If a service is needed by the example, but configuring it is not in the scope of the example, the assumption is that you have the knowledge to properly configure it, and only the resulting state of the service is described in the documentation.

Expert

Expert examples require the highest level of knowledge of the subject matter. You are expected to perform many tasks based on feature-based documentation and manuals, and the documentation is aimed at most complex scenarios.

APPENDIX I. GLOSSARY

I.1. PRODUCT AND PROJECT NAMES

Developer Launcher (developers.redhat.com/launch)

developers.redhat.com/launch called Developer Launcher is a stand-alone getting started experience provided by Red Hat. It helps you get started with cloud-native development on OpenShift. It contains functional example applications that you can download, build, and deploy on OpenShift.

Minishift or CDK

An OpenShift cluster running on your machine using Minishift.

I.2. TERMS SPECIFIC TO DEVELOPER LAUNCHER

Example

An application specification, for example *a web service with a REST API*.
Examples generally do not specify which language or platform they should run on; the description only contains the intended functionality.

Example application

A language-specific implementation of a particular example on a particular runtime. Example applications are listed in an examples catalog.
For example, an example application is a web service with a REST API implemented using the Thorntail runtime.

Examples Catalog

A Git repository that contains information about example applications.

Runtime

A platform that executes an example application. For example, Thorntail or Eclipse Vert.x.